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

Input:

2 -> 3 -> 3 -> 4

Output:

2 -> 3 -> 4

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

struct Node {
    int data;
    struct Node* next;
};
struct Node* head = NULL;


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

void insertNode()
{
    struct Node* temp;
    //creating new node
    temp = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter node data: ");
    scanf("%d",&temp->data);
    temp->next = NULL;
    if(head==NULL)
    {
        head = temp; //if list is empty, we return
            return;
    }
    else{
        struct Node* ptr = head;
        while(ptr->next!=NULL)
        {
            ptr = ptr->next;
        }
               // tail node pointing to new node
```

```
        ptr->next = temp;
    }
}

struct Node* removeDuplicates(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }

    struct Node* current = head;
    struct Node* next = NULL;

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

    return head;
}

// Function to print the linked list
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int n;
    printf("Enter the number of elements in the linked list: ");
    scanf("%d", &n);

    for(int i=0; i<n; i++){
        insertNode();
    }

    printf("Original linked list: ");
```

```
    printList(head);

    head = removeDuplicates(head);

    printf("Linked list after removing duplicates: ");
    printList(head);

    return 0;
}
```

OUTPUT:

```
C:\Users\Jyotsna\Downloads\C programs>a.exe
Enter the number of elements in the linked list: 5
Enter node data: 4
Enter node data: 6
Enter node data: 6
Enter node data: 3
Enter node data: 7
Original linked list: 4 -> 6 -> 6 -> 3 -> 7 -> NULL
Linked list after removing duplicates: 4 -> 6 -> 3 -> 7 -> NULL

C:\Users\Jyotsna\Downloads\C programs>a.exe
Enter the number of elements in the linked list: 4
Enter node data: 2
Enter node data: 3
Enter node data: 3
Enter node data: 4
Original linked list: 2 -> 3 -> 3 -> 4 -> NULL
Linked list after removing duplicates: 2 -> 3 -> 4 -> NULL
```

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

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

// Structure for a node in the doubly linked list
struct Node {
    char data;
    struct Node* prev;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the doubly linked list
void insertAtEnd(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
```

```
    newNode->prev = temp;
}

// Function to rotate the doubly linked list by N nodes
void rotateList(struct Node** head, int N) {
    if (*head == NULL || N <= 0) {
        return;
    }

    struct Node* temp = *head;
    int listLength = 1;

    // Find the length of the doubly linked list
    while (temp->next != NULL) {
        temp = temp->next;
        listLength++;
    }

    // Adjust N if it is greater than the length of the list
    N = N % listLength;

    // If N is 0 (or a multiple of the list length), no need to rotate
    if (N == 0) {
        return;
    }

    // Traverse to the (length - N)th node
    temp = *head;
    for (int i = 1; i < listLength - N; i++) {
        temp = temp->next;
    }

    // Update head and tail pointers
    struct Node* newHead = temp->next;
    struct Node* newTail = temp;
    struct Node* oldHead = *head;

    // Rotate the list
    newTail->next = NULL;
    newHead->prev = NULL;
    temp = newHead;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = oldHead;
```

```c
    oldHead->prev = temp;

    *head = newHead;
}

// Function to print the doubly linked list
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%c ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    char value;
    int n, N;

    printf("Enter the number of elements in the doubly linked list: ");
    scanf("%d", &n);
    getchar();

    printf("Enter the elements: ");
    for (int i = 0; i < n; i++) {
        scanf("%c", &value);
        insertAtEnd(&head, value);
        getchar();
    }

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

    printf("Enter the number of nodes to rotate: ");
    scanf("%d", &N);

    printf("Rotating the list by %d nodes...\n", N);
    rotateList(&head, N);
    printf("Doubly linked list after rotation: ");
    printList(head);

    return 0;
}
```

OUTPUT:

```
C:\Users\Jyotsna\Downloads\C programs>a.exe
Enter the number of elements in the doubly linked list: 5
Enter the elements: a b c d e
Original doubly linked list: a b c d e
Enter the number of nodes to rotate: 2
Rotating the list by 2 nodes...
Doubly linked list after rotation: d e a b c

C:\Users\Jyotsna\Downloads\C programs>a.exe
Enter the number of elements in the doubly linked list: 5
Enter the elements: a b c d e
Original doubly linked list: a b c d e
Enter the number of nodes to rotate: 4
Rotating the list by 4 nodes...
Doubly linked list after rotation: b c d e a
```

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

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

#define MAX_SIZE 100

struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

int isEmpty(struct Queue* q) {
    return (q->rear == -1);
}

int isFull(struct Queue* q) {
    return (q->rear == MAX_SIZE - 1);
}

void enqueue(struct 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;
```

```c
}

int dequeue(struct Queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}

void sortQueue(struct Queue* q) {
    int i, j, temp;
    for (i = q->front; i <= q->rear; i++) {
        for (j = i + 1; j <= q->rear; j++) {
            if (q->items[i] > q->items[j]) {
                temp = q->items[i];
                q->items[i] = q->items[j];
                q->items[j] = temp;
            }
        }
    }
}

void display(struct Queue* q) {
    int i;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue: ");
    for (i = q->front; i <= q->rear; i++) {
        printf("%d ", q->items[i]);
    }
    printf("\n");
}

int main() {
    struct Queue* q = createQueue();
    int n, value, i;
```

```
    printf("Enter the number of elements in the queue: ");
    scanf("%d", &n);

    printf("Enter the elements of the queue: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &value);
        enqueue(q, value);
    }

    printf("Original ");
    display(q);

    sortQueue(q);

    printf("Sorted ");
    display(q);

    return 0;
}
```

OUTPUT:

```
C:\Users\Jyotsna\Downloads\C programs>gcc sorting.c

C:\Users\Jyotsna\Downloads\C programs>a.exe
Enter the number of elements in the queue: 5
Enter the elements of the queue: 4
2
7
5
1
Original Queue: 4 2 7 5 1
Sorted Queue: 1 2 4 5 7
```

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

1. **enqueue**: It adds an element to the rear of the queue.
2. **dequeue**: It removes and returns the element from the front of the queue.
3. **peek**: It returns the element at the front of the queue without removing it.
4. **isEmpty**: It checks if the queue is empty. Returns true if the queue is empty, false otherwise.
5. **isFull**: It checks if the queue is full. Returns true if the queue is full, false otherwise.
6. **size**: It returns the number of elements currently present in the queue.

5. Reverse the given string using stack

Input: (string)

"LetsLearn"

Output: (string)

"nraeLsteL"

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

#define MAX_SIZE 100

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

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

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

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

```c
}

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

char pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack->items[stack->top--];
}

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

    // Push each character onto the stack
    for (int i = 0; i < len; i++) {
        push(stack, str[i]);
    }

    // Pop each character from the stack to reverse the string
    char* reversed = (char*)malloc((len + 1) * sizeof(char));
    int index = 0;
    while (!isEmpty(stack)) {
        reversed[index++] = pop(stack);
    }
    reversed[index] = '\0';

    return reversed;
}

int main() {
    char input[MAX_SIZE];
    printf("Enter a string: ");
    scanf("%s", input);

    char* reversed = reverseString(input);
    printf("Reversed string: %s\n", reversed);
```

```
    free(reversed);

    return 0;
}
```

OUTPUT:

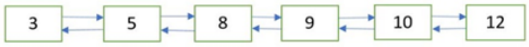
```
C:\Users\Jyotsna\Downloads\C programs>gcc revString.c

C:\Users\Jyotsna\Downloads\C programs>a.exe
Enter a string: LetsLearn
Reversed string: nraeLsteL
```

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

Doubly Linked List after insertion of 9




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

// Structure for a node in the doubly linked list
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node in sorted way into a sorted doubly linked list
void insertInSorted(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL || value < (*head)->data) {
        // Insert at the beginning
        newNode->next = *head;
        if (*head != NULL) {
            (*head)->prev = newNode;
        }
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL && temp->next->data < value) {
            temp = temp->next;
```

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

// Function to display the doubly linked list
void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

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

    // Input the number of elements in the sorted doubly linked list
    printf("Enter the number of elements in the sorted doubly linked list: ");
    scanf("%d", &n);

    // Input the elements of the sorted doubly linked list
    printf("Enter the elements of the sorted doubly linked list: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        insertInSorted(&head, value);
    }

    // Input the value to be inserted in sorted way
    printf("Enter the value to be inserted in sorted way: ");
    scanf("%d", &value);

    // Insert the value in sorted way
    insertInSorted(&head, value);

    // Display the updated sorted doubly linked list
    printf("Sorted doubly linked list after insertion: ");
    display(head);
```

```
    return 0;
}
```

OUTPUT:

```
C:\Users\Jyotsna\Downloads\C programs>a.exe
Enter the number of elements in the sorted doubly linked list: 5
Enter the elements of the sorted doubly linked list: 3
5
8
10
12
Enter the value to be inserted in sorted way: 9
Sorted doubly linked list after insertion: 3 5 8 9 10 12
```

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

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

#define MAX_SIZE 100

struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

struct Queue* createQueue() {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = -1;
    q->rear = -1;
    return q;
}

int isEmpty(struct Queue* q) {
    return (q->rear == -1);
}
```

```c
int isFull(struct Queue* q) {
    return (q->rear == MAX_SIZE - 1);
}

void enqueue(struct 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;
}

int dequeue(struct Queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}

int countElements(struct Queue* q) {
    if (isEmpty(q)) {
        return 0;
    }
    return q->rear - q->front + 1;
}

void display(struct Queue* q) {
    int i;
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements are: ");
```

```
    for (i = q->front; i <= q->rear; i++) {
        printf("%d ", q->items[i]);
    }
    printf("\n");
}

int main() {
    struct Queue* q = createQueue();
    int n, value;

    printf("Initialize a queue!\n");
    printf("Check the queue is empty or not? %s\n", isEmpty(q) ? "Yes" : "No");
    printf("Number of elements in queue: %d\n", countElements(q));

    printf("Insert some elements into the queue:\n");
    enqueue(q, 1);
    enqueue(q, 2);
    enqueue(q, 3);
    display(q);
    printf("Number of elements in queue: %d\n", countElements(q));

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

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

    return 0;
}
```

OUTPUT:

```
C:\Users\Jyotsna\Downloads\C programs>a.exe
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
```

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[]

```c
#include <stdio.h>

// Function to check if arr2 is a subset of arr1
int isSubset(int arr1[], int m, int arr2[], 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, m, arr2, 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, m, arr4, n))
        printf("arr4[] is a subset of arr3[]\n");
    else
        printf("arr4[] is not a subset of arr3[]\n");

    return 0;
}
```