ADVANCE

C PROGRAMMING

A. BUVANESHKUMAAR

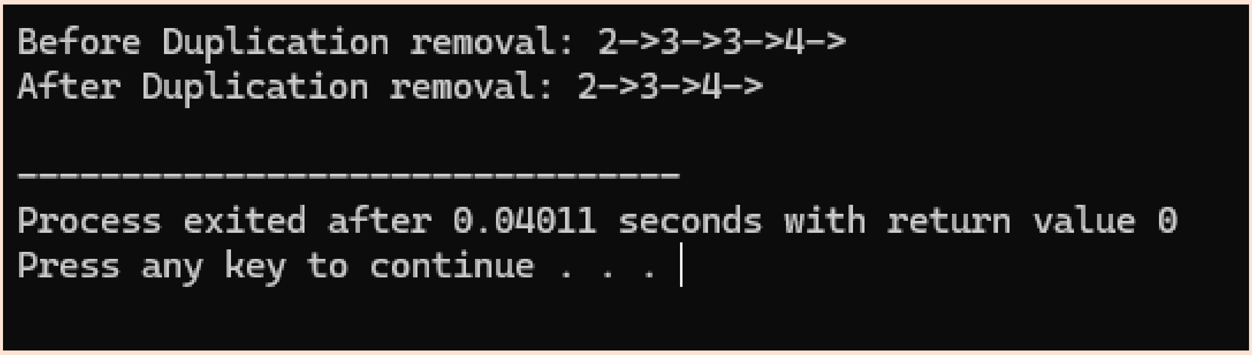
MEPCO SCHLENK ENGINEERING COLLEGE

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

Source code:

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

struct rnode
{
    int data;
    struct rnode* next;
};

struct rnode* insert(struct rnode* head, int value)
{
    struct rnode* nn = (struct rnode*)malloc(sizeof(struct rnode));
    nn->data = value;
    nn->next = NULL;

    if (head == NULL)
    {
        return nn;  // Return new node if the list is initially empty
    }

    struct rnode* now = head;

    while (now->next != NULL)
    {
        now = now->next;  // Move to the next node
    }

    now->next = nn;
          return head;
}

void removeDuplicates(struct rnode* head)
{
    struct rnode* now = head;

    while (now != NULL && now->next != NULL)
    {
        if (now->data == now->next->data)
        {
            struct rnode* duplicate = now->next;
            now->next = now->next->next;
            free(duplicate);
        }
        else
        {
            now = now->next;  // Move to the next node if no duplication
        }
    }
}

void display(struct rnode* head)
{
    struct rnode* now = head;

    while (now != NULL)
    {
        printf("%d->", now->data);
        now = now->next;
    }
    printf("\n");
}

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

    head = insert(head, 2);
    head = insert(head, 3);
    head = insert(head, 3);
    head = insert(head, 4);


    printf("Before Duplication removal: ");
    display(head);

    removeDuplicates(head);

    printf("After Duplication removal: ");
    display(head);

    // Free memory
    while (head != NULL)
    {
        struct rnode* temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

Result:



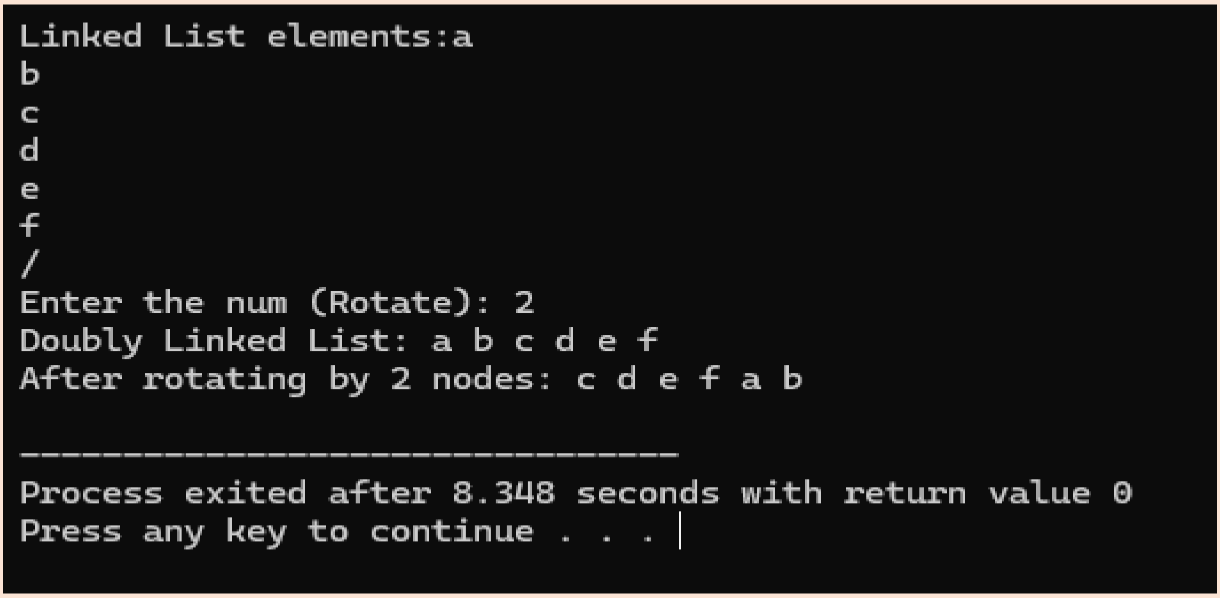
```
Before Duplication removal: 2->3->3->4->
After Duplication removal: 2->3->4->

--------------------------------
Process exited after 0.04011 seconds with return value 0
Press any key to continue . . .
```

2. Write a C program to rotate a doubly linked list by N nodes.

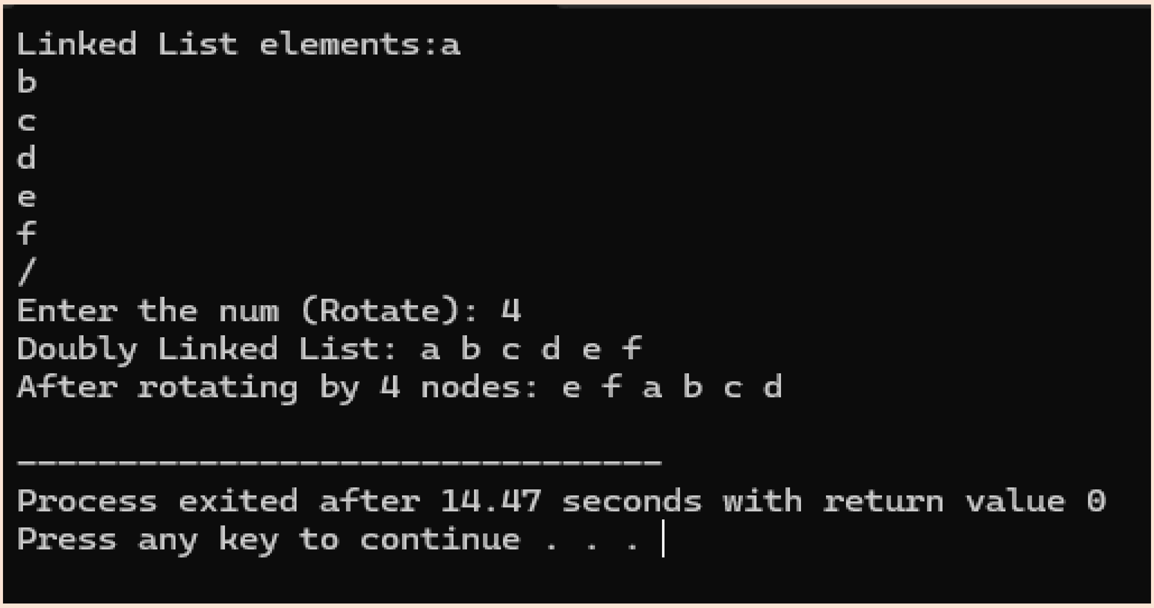
Source code:

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

struct node {
    char data;
    struct node* prev;
    struct node* nxt;
};

struct node* rotateDoublyList(struct node* head, int N) {
    if (head == NULL || N == 0) {
        return head;
    }

    struct node* lastnode = head;
    while (lastnode->nxt != NULL) {
        lastnode = lastnode->nxt;
    }
    int i =0;
    for (i = 0; i < N; ++i) {
        lastnode->nxt = head;
        head->prev = lastnode;
        head = head->nxt;
        head->prev = NULL;

        lastnode = lastnode->nxt;
        lastnode->nxt = NULL;
    }

    return head;
}

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

struct node* insert(struct node* head, char data) {
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = data;
    newnode->prev = NULL;
    newnode->nxt = NULL;

    if (head == NULL) {
        return newnode;
    }

    struct node* current = head;
    while (current->nxt != NULL) {
        current = current->nxt;
    }

    current->nxt = newnode;
    newnode->prev = current;
    return head;
}

int main() {
    struct node* head = NULL;
    char data;
    int N;

    printf("Linked List elements:");
    while (1) {
        scanf(" %c", &data);
        if (data == '/') {
            break;
        }
        head = insert(head, data);
    }

    printf("Enter the num (Rotate): ");
    scanf("%d", &N);

    printf("Doubly Linked List: ");
    printList(head);

    head = rotateDoublyList(head, N);

    printf("After rotating by %d nodes: ", N);
    printList(head);

    while (head != NULL) {
        struct node* temp = head;
        head = head->nxt;
        free(temp);
    }

    return 0;
}
```

Result:



```
Linked List elements:a
b
c
d
e
f
/
Enter the num (Rotate): 2
Doubly Linked List: a b c d e f
After rotating by 2 nodes: c d e f a b

--------------------------------
Process exited after 8.348 seconds with return value 0
Press any key to continue . . .
```



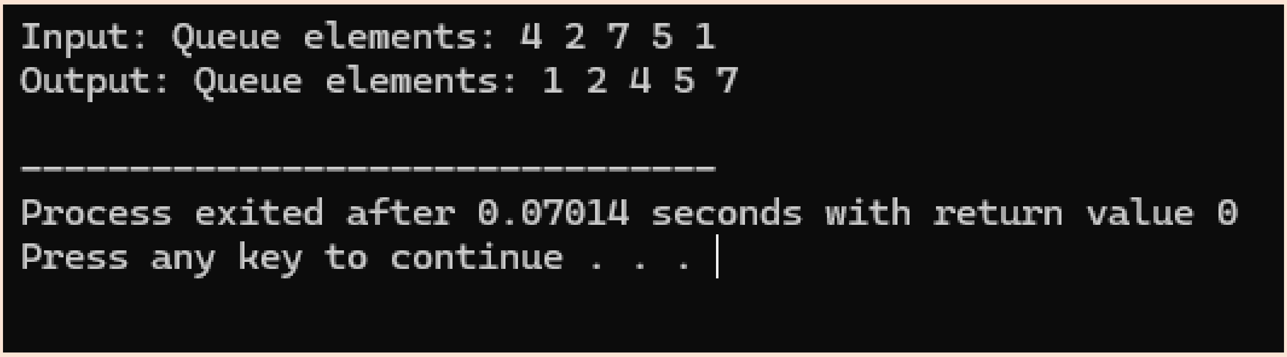
```
Linked List elements:a
b
c
d
e
f
/
Enter the num (Rotate): 4
Doubly Linked List: a b c d e f
After rotating by 4 nodes: e f a b c d

--------------------------------
Process exited after 14.47 seconds with return value 0
Press any key to continue . . .
```

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

Source code:

```
#include <stdio.h>
#define MAX_SIZE 100

struct Queue {
    int arr[MAX_SIZE];
    int front, rear;
};

void initQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

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

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

void enqueue(struct Queue* q, int data) {
    if (isFull(q)) {
        printf("Queue is full. Cannot enqueue element.\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
        q->rear = 0;
    } else {
        q->rear++;
    }
    q->arr[q->rear] = data;
}


int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Cannot dequeue element.\n");
        return -1;
    }
    int data = q->arr[q->front];
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front++;
    }
    return data;
}

void display(struct 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->arr[i]);
    }
    printf("\n");
}


void sortQueue(struct Queue* q) {
    int temp[MAX_SIZE];
    int n = q->rear - q->front + 1;


    for (int i = 0; i < n; i++) {
        temp[i] = dequeue(q); }
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (temp[j] > temp[j + 1]) {
                int tempVar = temp[j];
                temp[j] = temp[j + 1];
                temp[j + 1] = tempVar;
            }
        }
    }
    for (int i = 0; i < n; i++) {
        enqueue(q, temp[i]);
    }
}

int main() {
    struct Queue q;
    initQueue(&q);

   enqueue(&q, 4);
    enqueue(&q, 2);
    enqueue(&q, 7);
    enqueue(&q, 5);
    enqueue(&q, 1);

    printf("Input: ");
    display(&q);


    sortQueue(&q);

    printf("Output: ");
    display(&q);

    return 0;
}
```

Result:



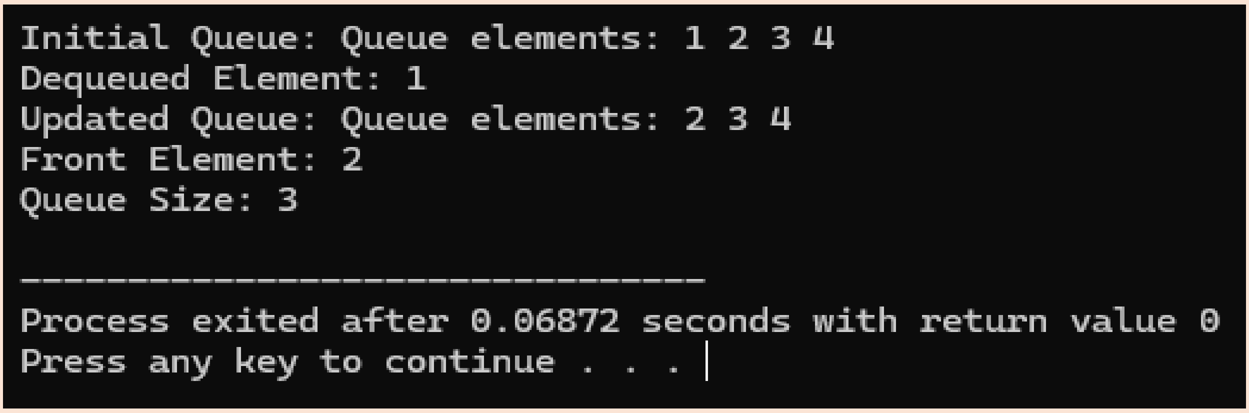
```
Input: Queue elements: 4 2 7 5 1
Output: Queue elements: 1 2 4 5 7

--------------------------------
Process exited after 0.07014 seconds with return value 0
Press any key to continue . . .
```

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

Source code:

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

#define MAX_SIZE 100

struct Queue {
    int arr[MAX_SIZE];
    int front, rear;
};

void initQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

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

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

void enqueue(struct Queue* q, int data) {
    if (isFull(q)) {
        printf("Queue is full. Cannot enqueue element.\n");
        return;
    }
    if (isEmpty(q)) {
        q->front = 0;
        q->rear = 0;
    } else {
        q->rear++;
    }
    q->arr[q->rear] = data;
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Cannot dequeue element.\n");
        return -1;
    }
    int data = q->arr[q->front];
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front++;
    }
    return data;
}

int peek(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Cannot peek element.\n");
        return -1;
    }
    return q->arr[q->front];
}

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


void display(struct 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->arr[i]);
    }
    printf("\n");
}

int main() {
    struct Queue q;
    initQueue(&q);
    enqueue(&q, 1);
    enqueue(&q, 2);
    enqueue(&q, 3);
    enqueue(&q, 4);

    printf("Initial Queue: ");
    display(&q);


    int dequeuedElement = dequeue(&q);
    printf("Dequeued Element: %d\n", dequeuedElement);


    printf("Updated Queue: ");
    display(&q);


    int frontElement = peek(&q);
    printf("Front Element: %d\n", frontElement);


    int queueSize = size(&q);
    printf("Queue Size: %d\n", queueSize);

    return 0;
}
```

Result:



```
Initial Queue: Queue elements: 1 2 3 4
Dequeued Element: 1
Updated Queue: Queue elements: 2 3 4
Front Element: 2
Queue Size: 3

--------------------------------
Process exited after 0.06872 seconds with return value 0
Press any key to continue . . .
```

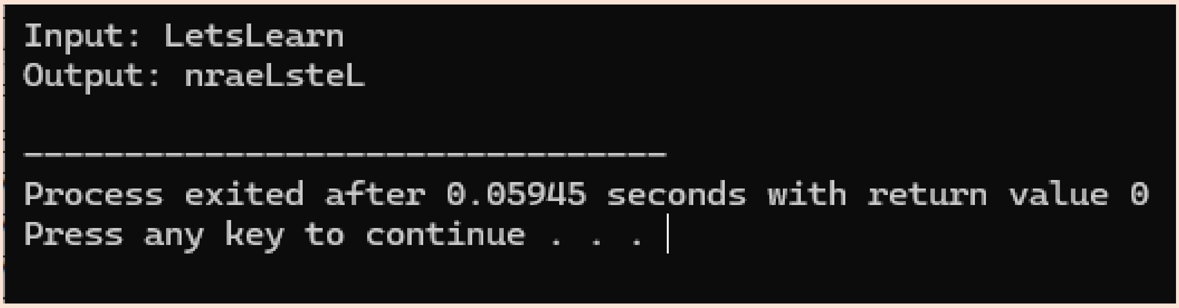
5. Reverse the given string using stack

Source code:

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

```
#define MAX_SIZE 100

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


void initStack(struct Stack* stack) {
    stack->top = -1;
}

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

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

void push(struct Stack* stack, char data) {
    if (isFull(stack)) {
        printf("Stack Overflow. Cannot push element '%c'.\n", data);
        return;
    }
    stack->arr[++stack->top] = data;
}

char pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow. Cannot pop element.\n");
        return '\0';
    }
    return stack->arr[stack->top--];
}

void reverseString(char* str) {
    int len = strlen(str);
    struct Stack stack;
    initStack(&stack);

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

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

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

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

    reverseString(str);

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

    return 0;
}
```

Result:



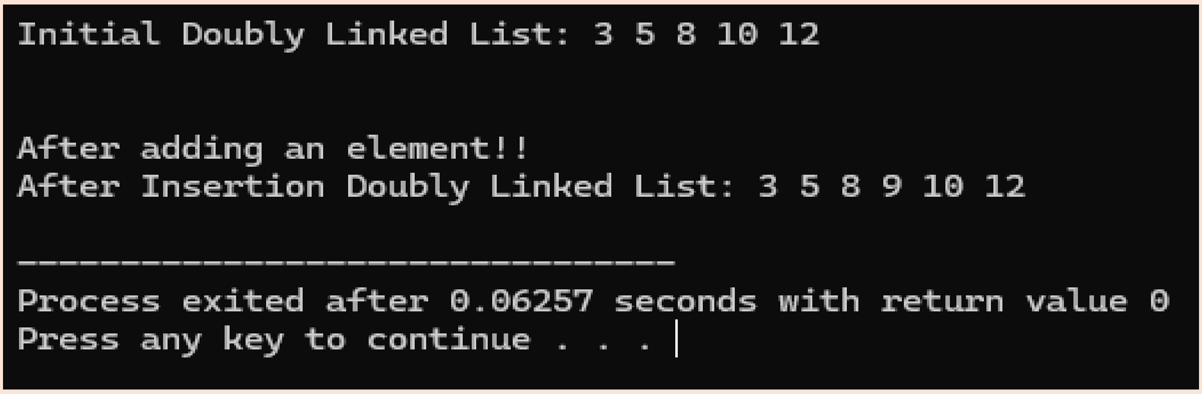
```
Input: LetsLearn
Output: nraeLsteL

--------------------------------
Process exited after 0.05945 seconds with return value 0
Press any key to continue . . .
```

6. Insert value in sorted way in a sorted doubly linked list

Source code:

```
#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));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}


void insertSorted(struct Node** head, int value) {
    struct Node* newNode = createNode(value);


    if (*head == NULL) {
        *head = newNode;
        return;
    }


    if (value < (*head)->data) {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
        return;
    }

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


    while (current != NULL && current->data < value) {
        prevNode = current;
        current = current->next;
    }


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


void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    printf("Doubly Linked List: ");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

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


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

    printf("Initial ");
    printList(head);
    puts("\n");
        puts("After adding an element!!");

    insertSorted(&head, 9);
    printf("After Insertion ");
    printList(head);

    return 0;
}
```

Result:



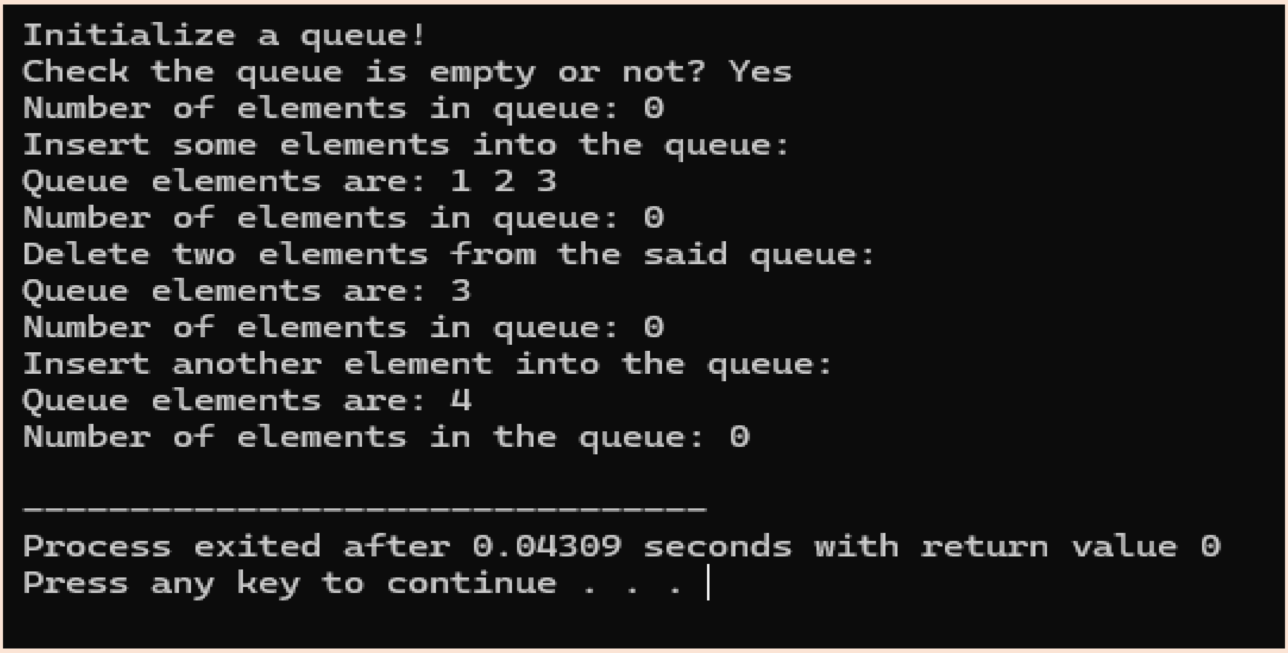
```
Initial Doubly Linked List: 3 5 8 10 12


After adding an element!!
After Insertion Doubly Linked List: 3 5 8 9 10 12

--------------------------------
Process exited after 0.06257 seconds with return value 0
Press any key to continue . . .
```

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

Source code:

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

#define MAX_SIZE 100

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

struct Queue* createQueue();
void enqueue(struct Queue* queue, int value);
int dequeue(struct Queue* queue);
int isEmpty(struct Queue* queue);
int size(struct Queue* queue);



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

void enqueue(struct Queue* queue, int value) {
    if (queue->rear == MAX_SIZE - 1) {
        printf("Queue is full!\n");
        return;
    }
    if (queue->front == -1)
        queue->front = 0;
    queue->rear++;
    queue->items[queue->rear] = value;
}

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

int isEmpty(struct Queue* queue) {
    if (queue->rear == -1)
        return 1;
    else
        return 0;
}

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

int main() {
    struct Queue* queue = createQueue();

    printf("Initialize a queue!\n");

    if (isEmpty(queue))
        printf("Check the queue is empty or not? Yes\n");
    else
        printf("Check the queue is empty or not? No\n");

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

    printf("Insert some elements into the queue:\n");
    enqueue(queue, 1);
    enqueue(queue, 2);
    enqueue(queue, 3);
    printf("Queue elements are: ");
    while (!isEmpty(queue)) {
        printf("%d ", dequeue(queue));
    }
    printf("\n");
    printf("Number of elements in queue: %d\n", size(queue));

    printf("Delete two elements from the said queue:\n");
    enqueue(queue, 3);
    printf("Queue elements are: ");
    while (!isEmpty(queue)) {
        printf("%d ", dequeue(queue));
    }
    printf("\n");
    printf("Number of elements in queue: %d\n", size(queue));

    printf("Insert another element into the queue:\n");
    enqueue(queue, 4);
    printf("Queue elements are: ");
    while (!isEmpty(queue)) {
        printf("%d ", dequeue(queue));
    }
    printf("\n");
    printf("Number of elements in the queue: %d\n", size(queue));

    return 0;
}
```

Result:



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

--------------------------------
Process exited after 0.04309 seconds with return value 0
Press any key to continue . . .
```

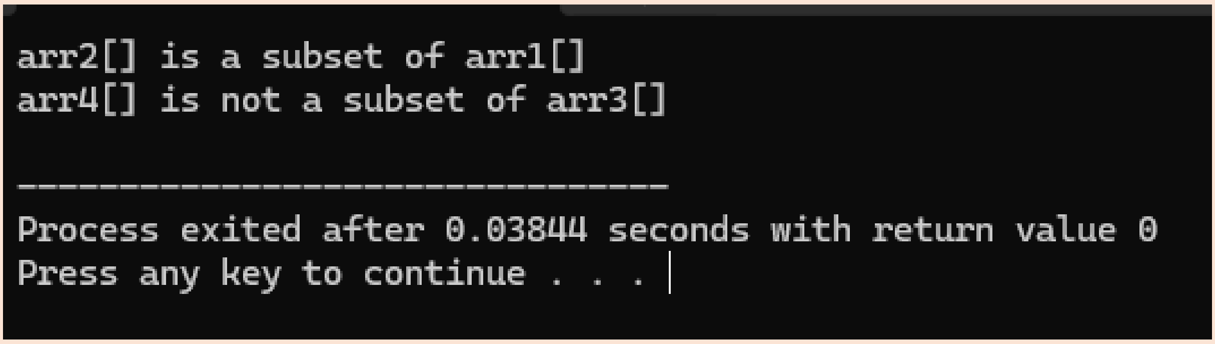
8. Write a C program to Find whether an array is a subset of another array.

Source code:

```
#include <stdio.h>

int isSubset(int arr1[], int m, int arr2[], int n);

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;
}

int isSubset(int arr1[], int m, int arr2[], int n) {
    int i, j;
    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;
}
```

Result:



```
arr2[] is a subset of arr1[]
arr4[] is not a subset of arr3[]

--------------------------------
Process exited after 0.03844 seconds with return value 0
Press any key to continue . . .
```

Thank you!