1 – Remove Duplicate

Code:

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

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

void insert(struct Node **head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* last = *head;
    new_node->data = data;
    new_node->next = NULL;
    if (*head == NULL) {
        *head = new_node;
        return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
}

void RemDup(struct Node *head) {
    struct Node *current = head, *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;
        }
    }
}

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

```
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    insert(&head, 2);
    insert(&head, 3);
    insert(&head, 3);
    insert(&head, 4);
    printf("Before removing duplicate:\n");
    print(head);
    RemDup(head);
    printf("After removing duplicate:\n");
    print(head);
    // Here you should ideally have a function to free the entire list to
prevent memory leaks.
    return 0;
}
```

Output –

```
Before removing duplicate:
 2 -> 3 -> 3 -> 4 NULL
After removing duplicate:
 2 -> 3 -> 4 NULL
```

2 – Rotate doubly linked list

Code:

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

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

void append(struct Node** head_ref, char 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) {
```

```
        new_node->prev = NULL;
        *head_ref = new_node;
        return;
    }
    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 *NthNode = current;
    while (current->next != NULL)
        current = current->next;
    current->next = *head_ref;
    (*head_ref)->prev = current;
    *head_ref = NthNode->next;
    (*head_ref)->prev = NULL;
    NthNode->next = NULL;
}

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

int main() {
    struct Node* head = NULL;
    int n, N;
    char elem;
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("Enter nodes: ");
    for (int i = 0; i < n; i++) {
        scanf(" %c", &elem);
        append(&head, elem);
    }
    printf("Enter N for rotation: ");
```

```
    scanf("%d", &N);
    rotate(&head, N);
    printList(head);
    return 0;
}
```

Output:

```
Enter number of nodes: 5
Enter nodes: a b c d e
Enter N for rotation: 2
c d e a b
```

3 – Sort Elements in queue

Code:

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

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

typedef struct Queue {
    Node *front, *rear;
} Queue;

void initializeQueue(Queue *q) {
    q->front = q->rear = NULL;
}

void enqueue(Queue *q, int value) {
    Node *newNode = (Node*)malloc(sizeof(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(Queue *q) {
    if (q->front == NULL)
```

```
        return INT_MIN;
    Node *temp = q->front;
    int data = temp->data;
    q->front = q->front->next;
    if (q->front == NULL)
        q->rear = NULL;
    free(temp);
    return data;
}

void sortQueue(Queue *q) {
    Node *i, *j;
    int temp;
    for(i = q->front; i != NULL; i = i->next) {
        for(j = i->next; j != NULL; j = j->next) {
            if(i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

void displayQueue(Queue *q) {
    Node *temp = q->front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    int n, data;
    initializeQueue(&q);

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

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

    sortQueue(&q);
```

```
    printf("Sorted Queue:\n");
    displayQueue(&q);

    while (q.front != NULL) {
        dequeue(&q);
    }

    return 0;
}
```

Output:

```
Enter number of elements: 5
Enter queue elements:
4 2 7 5 1
Sorted Queue:
1 2 4 5 7
```

4- All queue function

a) **enqueue**: Add an element to the end of the queue.

b) **dequeue**: Remove an element from the front of the queue.

c) **front**: Get the value of the front element without removing it.

d) **rear**: Get the value of the last element without removing it.

e) **isEmpty**: Check if the queue is empty.

f) **isFull**: Check if the queue is full (relevant for queue implementations with a fixed size, like array-based queues).

g) **size**: Get the number of elements currently in the queue.

h) **initializeQueue**: Set up the queue for use, usually by setting the front and rear pointers to **NULL**.

i) **displayQueue**: Print all elements of the queue.

j) **clearQueue**: Remove all elements from the queue and free memory if needed.

5 – Reverse string using Stack

Code:

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

#define MAX 100
```

```c
int top = -1;
char stack[MAX];

void push(char ch) {
    if(top == (MAX - 1)) {
        printf("Stack Overflow\n");
    } else {
        top = top + 1;
        stack[top] = ch;
    }
}

char pop() {
    if(top == -1) {
        printf("Stack Underflow\n");
        return -1;
    } else {
        char ch = stack[top];
        top = top - 1;
        return ch;
    }
}

void reverse(char str[]) {
    int n = strlen(str);

    for(int i = 0; i < n; i++)
        push(str[i]);

    for(int i = 0; i < n; i++)
        str[i] = pop();
}

int main() {
    char str[MAX];

    printf("Enter a string: ");
    fgets(str, MAX, stdin);
    str[strcspn(str, "\n")] = 0;

    reverse(str);

    printf("Reversed string is: %s\n", str);
    return 0;
}
```

Output:

```
Enter a string: LetsLearn
Reversed string is: nraeLsteL
```

6- Doubly Linked List sorted after insertion

Code –

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

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

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

void insertAtHead(Node** head, int data) {
    Node* newNode = getNewNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    (*head)->prev = newNode;
    newNode->next = *head;
    *head = newNode;
}

void sortedInsert(Node** head, int data) {
    Node* newNode = getNewNode(data);

    if (*head == NULL || (*head)->data >= newNode->data) {
        newNode->next = *head;
        if (*head != NULL) {
            (*head)->prev = newNode;
        }
        *head = newNode;
    } else {
        Node* current = *head;
        while (current->next != NULL && current->next->data < newNode->data) {
```

```
            current = current->next;
        }
        newNode->next = current->next;

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

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

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

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

    printf("Enter the elements in sorted order:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        sortedInsert(&head, data);
    }

    printf("Enter the value to insert: ");
    scanf("%d", &data);

    sortedInsert(&head, data);
    printf("Doubly Linked List after insertion of %d:\n", data);
    printList(head);

    return 0;
}
```

Output –

```
Enter the number of elements in the doubly linked list: 5
Enter the elements in sorted order:
3 5 8 10 9
Enter the value to insert: 12
Doubly Linked List after insertion of 12:
3 5 8 9 10 12
```

7- Queue Operations

Code –

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

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

typedef struct Queue {
    Node* front;
    Node* rear;
    int count;
} Queue;

void initializeQueue(Queue* q) {
    q->front = q->rear = NULL;
    q->count = 0;
}

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

void enqueue(Queue* q, int value) {
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = value;
    temp->next = NULL;

    if (q->rear == NULL) {
        q->front = q->rear = temp;
    } else {
        q->rear->next = temp;
        q->rear = temp;
    }
    q->count++;
}

int dequeue(Queue* q) {
    if (isEmpty(q)) {
```

```
        return INT_MIN;
    }
    Node* temp = q->front;
    int data = temp->data;
    q->front = q->front->next;

    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    q->count--;
    return data;
}

void showQueue(Queue* q) {
    Node* temp = q->front;
    while(temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void menu() {
    printf("Choose an operation:\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Show Queue\n");
    printf("4. Count of Elements\n");
    printf("5. Exit\n");
}

int main() {
    Queue q;
    int choice, value;

    initializeQueue(&q);
    menu();
    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
```

```
            case 2:
                value = dequeue(&q);
                if (value != INT_MIN) {
                    printf("Deleted value: %d\n", value);
                } else {
                    printf("Queue is empty.\n");
                }
                break;
            case 3:
                if (!isEmpty(&q)) {
                    printf("Queue elements are: ");
                    showQueue(&q);
                } else {
                    printf("Queue is empty.\n");
                }
                break;
            case 4:
                printf("Number of elements in queue: %d\n", q.count);
                break;
            case 5:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

Output-

```
Choose an operation:
1. Insert
2. Delete
3. Show Queue
4. Count of Elements
5. Exit
Enter your choice: 4
Number of elements in queue: 0
Enter your choice: 1
Enter value to insert: 1
Enter your choice: 1
Enter value to insert: 2
Enter your choice: 1
Enter value to insert: 3
Enter your choice: 3
Queue elements are: 1 2 3
Enter your choice: 4
Number of elements in queue: 3
Enter your choice: 2
Deleted value: 1
Enter your choice: 2
Deleted value: 2
Enter your choice: 3
Queue elements are: 3
Enter your choice: 4
Number of elements in queue: 1
Enter your choice: 1
Enter value to insert: 4
Enter your choice: 4
Number of elements in queue: 2
Enter your choice: 3
Queue elements are: 3 4
Enter your choice: 5
Exiting program.
```

8- Subset Array

Code:

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

int isSubset(int *arr1, int size1, int *arr2, int size2) {
    int i, j;
    for (i = 0; i < size2; i++) {
        for (j = 0; j < size1; j++) {
            if(arr2[i] == arr1[j])
                break;
        }
        if (j == size1)
            return 0;
    }
    return 1;
}

int main() {
    int *arr1, *arr2, size1, size2, i;

    printf("Enter size of array 1: ");
    scanf("%d", &size1);
    arr1 = (int *)malloc(size1 * sizeof(int));

    printf("Enter elements of array 1:\n");
    for(i = 0; i < size1; i++) {
        scanf("%d", &arr1[i]);
```

```
    }

    printf("Enter size of array 2: ");
    scanf("%d", &size2);
    arr2 = (int *)malloc(size2 * sizeof(int));

    printf("Enter elements of array 2:\n");
    for(i = 0; i < size2; i++) {
        scanf("%d", &arr2[i]);
    }

    if(isSubset(arr1, size1, arr2, size2))
        printf("arr2[] is a subset of arr1[]\n");
    else
        printf("arr2[] is not a subset of arr1[]\n");

    free(arr1);
    free(arr2);

    return 0;
}
```

Output:

```
Enter size of array 1: 6
Enter elements of array 1:
11 1 13 21 3 7
Enter size of array 2: 4
Enter elements of array 2:
11 3 7 1
arr2[] is a subset of arr1[]
```