Assignment 2

Ramana Srivats S

Anna University

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

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
   int val;
   struct Node* next;
};
typedef struct Node* Node;
int main()
{
   Node head = (Node)malloc(sizeof(Node));
   Node ptr = head;
   for (int i = 0; i < 5; i++)
   {
      ptr->val = i;
      if (i==4)
      {
         ptr->val = 3;
      }
      if (i!=4)
      {ptr->next = (Node)malloc(sizeof(Node));
      ptr = ptr->next;
      ptr->val = 0;}

   }
   ptr->next = NULL;
   ptr = head;
   while (ptr)
   {
      printf("\n%d",ptr->val);
      ptr = ptr->next;
   }
   ptr = head;
   Node nxt = head->next;
   int temp=ptr->val;
   while (nxt)
   {
      if (ptr->val == nxt->val)
```

```
        {
            if (nxt->next)
            {
                ptr->next = nxt->next;
                break;
            }
            else{
                ptr->next = NULL;
                break;
            }
        }
        ptr = ptr->next;
        nxt = nxt->next;
    }
    while (head)
    {
        printf("\n%d",head->val);
        head = head->next;
    }
    return 0;
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
   int val;
   struct Node* prev;
   struct Node* next;
};
struct Node* head = NULL;
struct Node* tail = NULL;
void addNode(int data)
{
   struct Node* node = (struct Node*)malloc(sizeof(struct Node));
   node->val = data;
   if (head==NULL)
```

```
    {
        head = tail = node;
        node->next = NULL;
        node->prev = NULL;
    }
    else{
        tail->next = node;
        node->prev = tail;
        tail = tail->next;
        node->next = NULL;
    }
}
void rotate(int N)
{
    tail->next = head;
    for (int i = 0; i < N; i++)
    {
        head = head->next;
        tail = tail->next;
    }
    head->prev = NULL;
    tail->next = NULL;
}
int main()
{
    for (int i=0; i<5; i++)
    {
        addNode(i);
    }
    struct Node* ptr = head;
    printf("Before rotating\n");
```

```
    while (ptr)
    {

       printf("%d=",ptr->val);
       ptr=ptr->next;
    }
    printf("\n");
    rotate(2);
    ptr = head;
    printf("After rotating\n");
    while (ptr)
    {
       printf("%d=",ptr->val);
       ptr=ptr->next;
    }
}
```

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

```
#include <stdio.h>
#define MAX 100
int queue[MAX];
int front = -1;
int back = -1;
void enque(int val)
{
   if (front == -1)
   {
      front=0;
   }
   if (back == MAX-1)
   {
      printf("Queue is full");
```

```
        return;
    }
    back++;
    queue[back] = val;
    return;
}
int dequeue()
{
    if (back==-1||front>back)
    {
        front = -1;
        printf("No elements in queue");
        return 0;
    }
    else{
        int num = queue[front];
        front = front + 1;
        return num;
    }
}
void sort_queue()
{
    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;
            }
```

```
        }
    }
}
void display()
{
    if (front==-1)
    {
        printf("No elements in the queue");
    }
    for (int i=front; i <= back; i++ )
    {
        printf("\n%d",queue[i]);
    }
}
int main()
{
    enque(4);
    enque(2);
    enque(7);
    enque(5);
    enque(1);
    sort_queue();
    display();
    enque(3);
    sort_queue();
    display();
}
```

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

   enqueue() – Used to add elements to the queue

   dequeue() – Used to remove elements from the queue

isEmpty() – Used to check whether the queue is empty

isFull() – Used to check whether the queue is full

5. Reverse the given string using stack

Input: (string)

"LetsLearn"

Output: (string)

"nraeLsteL"

```
#include<string.h>
#include<stdio.h>
#define size 50
char stack[size];
int back=-1;
void push(char val)
{
   if (back==size-1)
   {
      printf("\nStack is full");
      return;
   }
   else{
      back++;
      stack[back] = val;
   }
}
char pop()
{
   if (back==-1)
   {
```

```
        printf("\nStack is empty");
        return "-1";
    }
    else{
        int num = stack[back];
        back = back-1;
        return num;
    }
}
int main()
{
    char str1[20] = "LetsLearn";
    for (int i=0; i < strlen(str1);i++)
    {
        push(str1[i]);
    }
    int i = 0;
    while (back>=0)
    {
        printf("%c",pop());
    }
}
```

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.

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int val;
    struct Node* prev;
```

```
    struct Node* next;
};
struct Node* head = NULL;
struct Node* tail = NULL;
struct Node* ptr;
struct Node* previous = NULL;

void addNode(int data)
{
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->val = data;
    node->next = NULL;
    node->prev = NULL;

    if (head==NULL)
    {
        head = tail = node;
    }
    else{
        ptr = head;
        while (ptr != NULL && ptr->val < data)
        {
            ptr = ptr->next;
        }
        if (ptr == NULL)
        {
            tail->next = node;
            node->prev = tail;
            tail = node;
        }
        else if (ptr->prev == NULL)
```

```
        {
            node->next = head;
            head->prev = node;
            head = node;
        } else {
            node->prev = ptr->prev;
            node->next = ptr;
            ptr->prev->next = node;
            ptr->prev = node;
        }
    }
}

void display()
{
    ptr = head;
    printf("\n");
    if (ptr==NULL)
    {
        printf("Doubly linked list is empty");
        return;
    }

    while(ptr)
    {
        printf("%d=",ptr->val);
        ptr = ptr->next;
    }
}

int main()
```

```
{
   addNode(3);
   addNode(5);
   addNode(10);
   addNode(8);
   addNode(12);
   display();
   addNode(1);
   display();
}
```

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

```
#include <stdio.h>
#define MAX 100
int queue[MAX];
int front = -1;
int back = -1;
void enque(int val)
{
   if (front == -1)
   {
      printf("\nInitializing the queue");
      front=0;
   }
   if (back == MAX-1)
   {
      printf("Queue is full");
      return;
   }
```

```
    back++;
    queue[back] = val;
    return;
}
int dequeue()
{
    if (back==-1||front>back)
    {
        front = -1;
        printf("No elements in queue");
        return 0;
    }
    else{
        int num = queue[front];
        front = front + 1;
        return num;
    }
}
void display()
{
    if (front==-1)
    {
        printf("\nNo elements in the queue");
        return;
    }
    printf("\nQueue elements are ");
    for (int i=front; i <= back; i++ )
    {
        printf("%d ",queue[i]);
    }
}
```

```
void QueueSize()
{
   if (front == -1)
   {
      printf("\nThe queue is empty");
      return;
   }
   else{
      printf("\nNumber of elements in queue: %d",back-front+1);
   }
}
int main()
{
   enque(2);
   enque(4);
   enque(3);
   QueueSize();
   display();
   dequeue();
   QueueSize();
   display();
}
```

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

```
#include <stdio.h>
int isSubset(int arr1[],int arr2[],int s1,int s2)
{
   int count = 0;
   for (int i = 0; i < s1;i++)
   {
```

```
        for (int j = i; j < s2; j++)
        {
            if (arr1[i] == arr2[j])
            {
                count = count + 1;
                break;
            }
        }
    }
    if (count==s1 || count==s2)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
int main()
{
    int arr1[] = {1,2,3,4,5};
    int arr2[] = {1,2,6};
    int s1 = sizeof(arr1)/sizeof(int);
    int s2 = sizeof(arr2)/sizeof(int);
    if (isSubset(arr1,arr2,s1,s2))
    {
        printf("\nArray 2 is a subset of Array 1");
    }
    else{
        printf("\nArray 2 is not a subset of Array 1");}
}
```