CN Lab -ARIF

1. Implement the data link layer framing methods such as character, character-stuffing and bit stuffing.

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

void characterStuffing(char* data, char* sd) {
    int n = strlen(data);
    int sl = 0;

    // Append DLE STX at the beginning
    sd[sl++] = 'D';
    sd[sl++] = 'L';
    sd[sl++] = 'E';
    sd[sl++] = 'S';
    sd[sl++] = 'T';
    sd[sl++] = 'X';

    for (int i = 0; i < n; i++) {
        if (data[i] == 'D' && data[i + 1] == 'L' && data[i + 2] == 'E') {
            // Insert another DLE after encountering "DLE" in the original data
            sd[sl++] = 'D';
            sd[sl++] = 'L';
            sd[sl++] = 'E';
        }
        sd[sl++] = data[i];
    }

    // Transmit DLE ETX at the end
    sd[sl++] = 'D';
    sd[sl++] = 'L';
    sd[sl++] = 'E';
    sd[sl++] = 'E';
    sd[sl++] = 'T';
    sd[sl++] = 'X';
    sd[sl] = '\0';
}

void characterDestuffing(char* sd, char* data) {
    int sl = strlen(sd);
    int n = 0;

    // Neglect initial DLE STX
    int i = 6; // Skip DLE STX

    while (i < sl - 6) { // Ignore DLE ETX at the end
        if (sd[i] == 'D' && sd[i + 1] == 'L' && sd[i + 2] == 'E') {
            // If DLE is present

                // Copy the same to output
                data[n++] = 'D';
                data[n++] = 'L';
                data[n++] = 'E';
                i += 6; // Skip the next DLE

        } else {
```

```
            data[n++] = sd[i];
            i++;
        }
    }

    data[n] = '\0';
}

int main() {
    char data[50];
    char sd[100];
    char desd[100];
    printf("Enter string: ");
    scanf("%s",&data);

    characterStuffing(data, sd);
    printf("Stuffed Data: %s\n", sd);

    characterDestuffing(sd, desd);
    printf("Destuffed Data: %s\n", desd);

    return 0;
}
```

1b

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

void bitStuffing(char* dat, char* stf) {
    int n = strlen(dat);
    int p = 0,cnt=0;
   for(int i=0;i<n;i++){
    stf[p++]=dat[i];
     if(dat[i]=='1'){
        cnt++;

        if(cnt==5){
         stf[p++]='0';
         cnt=0;
        }
      else{
         cnt=0;
      }

   }

   stf[p++]='\0';


}

void bitDestuffing(char* dat, char* dstf) {
 int n = strlen(dat);
    int p = 0,cnt=0;
   for(int i=0;i<n;i++){
    dstf[p++]=dat[i];
     if(dat[i]=='1'){
        cnt++;
```

```
        if(cnt==5){
         i++;
         cnt=0;
        }
     }
     else{
        cnt=0;
     }

   }

   dstf[p++]='\0';
}

int main() {
    char data[] = "1011111101"; // Example bit stream
    char stuffedData[100];
    char destuffedData[100];

    bitStuffing(data, stuffedData);
    printf("Stuffed Datas: %s\n", stuffedData);

    bitDestuffing(stuffedData, destuffedData);
    printf("Destuffed Data: %s\n", destuffedData);

    return 0;
}
```

2. Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC, CCIP.

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

unsigned int crc12(const char *data) {
    unsigned int crc = 0;
    unsigned int poly = 0x180F; // CRC-12 polynomial
    while (*data) {
        crc ^= (*data++ << 4);
        for (int i = 0; i < 8; i++) {
            if (crc & 0x800) {
                crc = (crc << 1) ^ poly;
            } else {
                crc <<= 1;
            }
        }
    }
    return crc & 0xFFF; // 12-bit CRC
}

unsigned int crc16(const char *data) {
    unsigned int crc = 0;
    unsigned int poly = 0x8005; // CRC-16 polynomial
    while (*data) {
        crc ^= (*data++ << 8);
        for (int i = 0; i < 8; i++) {
            if (crc & 0x8000) {
                crc = (crc << 1) ^ poly;
```

```
            } else {
                crc <<= 1;
            }
        }
    }
    return crc & 0xFFFF; // 16-bit CRC
}

unsigned int crcCCIP(const char *data) {
    unsigned int crc = 0;
    unsigned int poly = 0x1021; // CRC-CCIP polynomial
    while (*data) {
        crc ^= (*data++ << 8);
        for (int i = 0; i < 8; i++) {
            if (crc & 0x8000) {
                crc = (crc << 1) ^ poly;
            } else {
                crc <<= 1;
            }
        }
    }
    return crc & 0xFFFF; // 16-bit CRC
}

int main() {
    char data[] = "123456789"; // Example data

    printf("CRC-12: %X\n", crc12(data));
    printf("CRC-16: %X\n", crc16(data));
    printf("CRC-CCIP: %X\n", crcCCIP(data));

    return 0;
}
```

3. Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism.

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

int main() {
    int w, i, f, frames[50];
    bool ack[50]={false}; // Array to keep track of acknowledgments

    printf("Enter window size: ");
    scanf("%d", &w);
    printf("\nEnter number of frames to transmit: ");
    scanf("%d", &f);
    printf("\nEnter %d frames: ", f);
    for (i = 0; i < f; i++)
        scanf("%d", &frames[i]);

    printf("\nWith sliding window protocol and Go-Back-N ARQ, the frames will be sent in
          "(assuming no corruption of frames):\n\n");

    int sent=0;
    while(sent<f){
```

```
        for(i=sent;i<sent+w && i<f;i++){
           printf("Sending frame %d\n", frames[i]);
        }

         for(i=sent;i<sent+w && i<f;i++){
           if (rand() % 2) { // Randomly decide if ACK is received
        printf("Acknowledgment received for frame %d\n", frames[i]);
        ack[i]=true;
    } else {
        printf("Acknowledgment lost for frame %d\n", frames[i]);
        break; // Go-Back-N: Stop and resend from this frame
    }
        }

         for(i=sent;i<sent+w && i<f;i++){
          if(ack[i])
            sent++;
            else
            break;
        }
    }
    printf("\nAll frames sent and acknowledged.\n");
    return 0;
}
```

4. Implement Dijsktra's algorithm to compute the shortest path through a network

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

#define V 9 // Number of vertices in the graph

int findmin(int dist[], bool sptSet[]) {
    int min = INT_MAX, min_index;

    for(int i=0;i<V;i++){
        if(!sptSet[i]&& dist[i]<min){
            min=dist[i];
            min_index=i;
        }
    }

    return min_index;
}

void printSolution(int dist[]) {
    printf("Vertex \t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int sc) {
   int dist[V];
   bool sptset[V]={false};
   for (int i = 0; i < V; i++) {
    dist[i] = INT_MAX;
}
```

```
   dist[sc]=0;


   for(int i=0;i<V;i++){

    int u=findmin(dist,sptset);
    sptset[u]=true;

    for(int j=0;j<V;j++){
       if(!sptset[j]&& graph[u][j]&& dist[u]+graph[u][j]<dist[j]){
          dist[j]=dist[u]+graph[u][j];
       }
    }
   }
   printSolution(dist);

}
```

```
int main() {
    // Example graph
    int graph[V][V] = {
        {0, 4, 0, 0, 0, 0, 0, 8, 0},
        {4, 0, 8, 0, 0, 0, 0, 11, 0},
        {0, 8, 0, 7, 0, 4, 0, 0, 2},
        {0, 0, 7, 0, 9, 14, 0, 0, 0},
        {0, 0, 0, 9, 0, 10, 0, 0, 0},
        {0, 0, 4, 14, 10, 0, 2, 0, 0},
        {0, 0, 0, 0, 0, 2, 0, 1, 6},
        {8, 11, 0, 0, 0, 0, 1, 0, 7},
        {0, 0, 2, 0, 0, 0, 6, 7, 0}
    };

    dijkstra(graph, 0);

    return 0;
}
```

5.Take an example subnet of hosts and obtain a broadcast tree for the subnet.

```
#include <stdio.h>

int a[10][10], n;

void broadcast_tree(int root);

int main() {
    int i, j, root;

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

    printf("Enter adjacency matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            printf("Enter connection of %d>%d: ", i, j);
```

```
            scanf("%d", &a[i][j]);
        }
    }

    printf("Enter root node: ");
    scanf("%d", &root);

    broadcast_tree(root);

    return 0;
}

void broadcast_tree(int root) {
    int visited[10] = {0};
    int queue[10], front = -1, rear = -1;
    int i, j;

    printf("Broadcast Tree starting from node %d:\n", root);
    queue[++rear] = root;
    visited[root] = 1;

    while (front != rear) {
        int current = queue[++front];
        printf("%d ", current);

        for (i = 1; i <= n; i++) {
            if (a[current][i] == 1 && !visited[i]) {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}
```

6 Implement distance vector routing algorithm for obtaining routing tables at each node

```
#include <stdio.h>

#define infinity 999
#define v 5

struct node{
    int dist[v];
    int from[v];
} rt[v];

int main() {
    int adj[v][v];
    int n;
    printf("Enter no of vertices:");
    scanf("%d",&n);
    printf("Enter the cost matrix:\n");
       for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
            adj[i][i]=0;
            rt[i].dist[j]=adj[i][j];
            rt[i].from[j]=j;
```

```
        }
    }
int cnt;

    do{
         cnt=0;

        for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for(int k=0;k<n;k++){
                if(rt[i].dist[j]>adj[i][k]+rt[k].dist[j]){
                   rt[i].dist[j]=adj[i][k]+rt[k].dist[j] ;
                   rt[i].from[j]=k;
                   cnt++;
                }
            }


        }
    }while(cnt!=0);


    for (int i = 0; i < n; i++) {
         printf("\n\nState value for router %d is\n", i+1);
        for (int j = 0; j < n; j++) {
             printf("node %d via %d Distance %d\n", j + 1, rt[i].from[j] + 1, rt[i].dist

        }
    }



    return 0;
}
```

7. Implement data encryption and data decryption.

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

void encrypt(char *message, int key) {
    for (int i = 0; message[i] != '\0'; i++) {
        char ch = message[i];
        if (ch >= 'a' && ch <= 'z') {
            ch = ch + key;
            if (ch > 'z') {
                ch = ch - 'z' + 'a' - 1;
            }
            message[i] = ch;
        } else if (ch >= 'A' && ch <= 'Z') {
            ch = ch + key;
            if (ch > 'Z') {
                ch = ch - 'Z' + 'A' - 1;
            }
            message[i] = ch;
        }
```

```
    }
}

void decrypt(char *message, int key) {
    for (int i = 0; message[i] != '\0'; i++) {
        char ch = message[i];
        if (ch >= 'a' && ch <= 'z') {
            ch = ch - key;
            if (ch < 'a') {
                ch = ch + 'z' - 'a' + 1;
            }
            message[i] = ch;
        } else if (ch >= 'A' && ch <= 'Z') {
            ch = ch - key;
            if (ch < 'A') {
                ch = ch + 'Z' - 'A' + 1;
            }
            message[i] = ch;
        }
    }
}

int main() {
    char message[100];
    int key, choice;

    printf("Enter a message: ");
    gets(message);

    printf("Enter key: ");
    scanf("%d", &key);

    printf("Choose an option:\n1. Encrypt\n2. Decrypt\n");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            encrypt(message, key);
            printf("Encrypted message: %s\n", message);
            break;
        case 2:
            decrypt(message, key);
            printf("Decrypted message: %s\n", message);
            break;
        default:
            printf("Invalid choice\n");
    }

    return 0;
}
```

8. Write a program for congestion control using Leaky bucket algorithm

```
#include <stdio.h>

int main() {
    int b_s, o_r, ps, cont = 0;
    int n;
```

```c
    printf("Enter the bucket size: ");
    scanf("%d", &b_s);

    printf("Enter the output rate: ");
    scanf("%d", &o_r);

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

    for (int i = 0; i < n; i++) {
        printf("Enter the size of packet %d: ", i + 1);
        scanf("%d", &ps);

        if (ps <= (b_s - cont)) {
            cont += ps;
            printf("Packet %d added to the bucket. Current bucket content: %d bytes\n", i
        } else {
            printf("Packet %d discarded. bucket overflow.\n", i + 1);
        }

        cont -= o_r;
        if (cont < 0) {
            cont = 0;
        }
        printf("After output, bucket content: %d bytes\n", cont);
    }

    while (cont > 0) {
        cont -= o_r;
        if (cont < 0) {
            cont = 0;
        }
        printf("After output, bucket content: %d bytes\n", cont);
    }

    return 0;
}
```

9. Write a program for frame sorting technique used in buffers

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

int fnum[20]; // Frame sequence numbers
char finfo[20][20]; // Frame information
int n; // Number of frames

void sort() {
    int i, j, temp;
    char temp1[20];

    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            if (fnum[j]<fnum[i]) {
                // Swap sequence numbers
                temp = fnum[i];
```

```
                fnum[i] = fnum[j];
                fnum[j] = temp;

                // Swap frame information
                strcpy(temp1, finfo[i]);
                strcpy(finfo[i], finfo[j]);
                strcpy(finfo[j], temp1);
            }
        }
    }
}

int main() {
    int i;

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

    for (i = 0; i < n; i++) {
        printf("Enter sequence number for frame %d: ", i + 1);
       scanf("%d",&fnum[i]);
        printf("Enter the frame contents for sequence number %d: ", fnum[i]);
        scanf("%s", &finfo[i]);
    }

    sort();

    printf("\nThe frames in sequence are:\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%s\n", fnum[i], finfo[i]);
    }

    return 0;
}
```