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

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

void character_stuffing(char* data,char flag, char escape,char* stuffed);
void character_unstuffing(char* stuffed,char flag, char escape,char* unstuffed);
void bit_stuffing(char *data,char* stuffed);
void bit_unstuffing(char *stuffed,char* unstuffed);
void character_count_framing(char* data,char* framed);
void character_count_deframing(char* framed,char* deframed);

int main(){
   char data[] = "venky|bkl";
   char flag = '|';
   char escape = '/';
   char stuffed[256],unstuffed[256];
   printf("Character Stuffing\n");
   character_stuffing(data,flag,escape,stuffed);
   printf("Character stuffed %s\n",stuffed);
   character_unstuffing(stuffed,flag,escape,unstuffed);
   printf("Character unstuffed %s\n\n",unstuffed);

   char binary_data[] = "111110";
   char bit_stuffed[256],bit_unstuffed[256];

   printf("Bit Stuffing\n");
   bit_stuffing(binary_data,bit_stuffed);
   printf("Bit stuffed %s\n",bit_stuffed);
   bit_unstuffing(bit_stuffed,bit_unstuffed);
```

```
    printf("Bit unstuffed %s\n\n",bit_unstuffed);


    char count_data[]="BLANKSPACE";
    char framed[256],deframed[256];
    character_count_framing(count_data,framed);
    printf("Character count\n");
    printf("Character count framing : %s\n",framed);
    character_count_deframing(framed,deframed);
    printf("Character count deframing : %s\n",deframed);
    return 0;
}
void character_stuffing(char* data,char flag, char escape,char* stuffed){
    int j = 0;
    stuffed[j++] = flag;
    for(int i = 0;data[i]!= '\0';i++){
        if(data[i] == escape || data[i]== flag){
            stuffed[j++] = escape;
        }
        stuffed[j++] = data[i];
    }
    stuffed[j++] = flag;
    stuffed[j]= '\0';
}
void character_unstuffing(char* stuffed,char flag,char escape,char* unstuffed){
    int j =0;
    for(int i =0;stuffed[i]!='\0';i++){
        if(stuffed[i]== flag || stuffed[i] == escape){
            i++;
        }
        unstuffed[j++] = stuffed[i];
    }
```

```
    unstuffed[j]='\0';
}
void bit_stuffing(char *data,char* stuffed){
    int count = 0;
    int j =0;
    for(int i = 0;data[i]!= '\0';i++){
        stuffed[j++] = data[i];
        if(data[i]=='1'){
            count++;
            if(count == 5){
                stuffed[j++] = '0';
                count=0;
            }
        }else{
            count = 0;
        }
    }
    stuffed[j] = '\0';
}
void bit_unstuffing(char *stuffed,char *unstuffed){
    int count = 0;
    int j = 0;
    for(int i = 0;stuffed[i]!='\0';i++){
        unstuffed[j++] = stuffed[i];
        if(stuffed[i] == '1'){
            count++;
            if(count==5){
                i++;
                count=0;
            }
        }else{
```

```
            count = 0;
        }
    }
    unstuffed[j] = '\0';
}
void character_count_framing(char* data,char* framed){
    int length = strlen(data);
    sprintf(framed,"%02d%s",length,data);
}
void character_count_deframing(char* framed,char* deframed){
    int length;
    sscanf(framed,"%02d",&length);
    strncpy(deframed,framed+2,length);
    deframed[length] = '\0';
}
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h> // Include this header for uint16_t and uint8_t

#define POLY 0x80F

uint16_t crc_12(uint8_t data[], size_t length) {
    uint16_t crc = 0xFFF;
    for (size_t i = 0; i < length; i++) {
        crc ^= (data[i] << 4);
        for (size_t j = 0; j < 8; j++) {
            if (crc & 0x800) {
                crc = (crc << 1) ^ POLY;
            } else {
```

```
            crc <<= 1;
          }
        }
    }
    return crc & 0xFFF;
}
int main() {
    uint8_t data[] = {0x12, 0x34, 0x56};
    size_t length = sizeof(data) / sizeof(data[0]);

    uint16_t result = crc_12(data, length);

    printf("CRC-12 : 0x%03X\n", result); // Added newline for better output formatting
    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<time.h>
#include<stdlib.h>
#include<stdbool.h>

#define Window_size 4
#define loss_probablity 30
#define Total_frames 10

void sliding_window_protocol();
bool send_frame(int);
bool receive_ack(int);
```

```
int main(){
    srand(time(NULL));

    printf("Initializing Sliding Window with GO Back N protocol...\n");
    sliding_window_protocol();

    return 0;
}

void sliding_window_protocol(){
    int base = 0;
    int acknowledgements = 0;
    int next_frame = 0;

    while(acknowledgements < Total_frames){
        while(next_frame < base + Window_size && next_frame < Total_frames){
            if(send_frame(next_frame)){
                next_frame++;
            } else {
                break;
            }
        }

        for(int i = base; i < next_frame; i++){
            if(receive_ack(i)){
                base++;
                acknowledgements++;
            } else {
                printf("Retransmitting the frame %d for being lost acknowledgment\n", base);
                next_frame = base;
```

```
            break;
          }
        }
    }
    printf("\nALL FRAMES SENT SUCCESSFULLY\n");
}

bool send_frame(int frame){
    int random = rand() % 100;
    if(random < loss_probablity){
        printf("Frame %d lost during transmission\n", frame);
        return false;
    }
    printf("Frame %d sent successfully\n", frame);
    return true;
}

bool receive_ack(int frame){
    int random = rand() % 100;
    if(random < loss_probablity){
        printf("Frame %d acknowledgment is lost\n", frame);
        return false;
    }
    printf("Frame %d acknowledged successfully\n", frame);
    return true;
}
```

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

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

```
#define NUM_NODES 5

int graph[NUM_NODES][NUM_NODES] = {
   {0, 10, 0, 30, 100},
   {10, 0, 50, 0, 0},
   {0, 50, 0, 20, 10},
   {30, 0, 20, 0, 60},
   {100, 0, 10, 60, 0}
};

void dijkstra(int);
int find_min_distance(int[], bool[]);

int main(){
   int start = 0;
   dijkstra(start);
   return 0;
}

void dijkstra(int start){
   int distance[NUM_NODES];
   bool visited[NUM_NODES];

   for(int i = 0; i < NUM_NODES; i++){
      distance[i] = INT_MAX;
      visited[i] = false;
   }
   distance[start] = 0;

   for(int count = 0; count < NUM_NODES - 1; count++){
```

```
        int u = find_min_distance(distance, visited);

        visited[u] = true;

        for(int v = 0; v < NUM_NODES; v++){
            if(!visited[v] && graph[u][v] && distance[u] != INT_MAX && distance[u] + graph[u][v] < distance[v]){
                distance[v] = distance[u] + graph[u][v];
            }
        }
    }

    printf("NODE\tDISTANCE FROM SOURCE\n");
    for(int i = 0; i < NUM_NODES; i++){
        printf("%d\t%d\n", i, distance[i]);
    }
}

int find_min_distance(int distance[], bool visited[]){
    int min = INT_MAX, min_index = -1;
    for(int i = 0; i < NUM_NODES; i++){
        if(!visited[i] && distance[i] <= min){
            min = distance[i];
            min_index = i;
        }
    }
    return min_index;
}
```

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

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

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

#define MAX_NODES 5
#define INF INT_MAX

int graph[MAX_NODES][MAX_NODES] = {
   {0, 1, 3, INF, INF},
   {1, 0, INF, 4, 5},
   {3, INF, 0, INF, 6},
   {INF, 4, INF, 0, 2},
   {INF, 5, 6, 2, 0}
};

void prims_broadcast_tree();

int main(){
   printf("Broadcasting Tree using Prim's Algorithm\n");
   prims_broadcast_tree();
   return 0;
}

void prims_broadcast_tree(){
   int parent[MAX_NODES];
   int weight[MAX_NODES];
   bool in_tree[MAX_NODES];

   for(int i = 0; i < MAX_NODES; i++){
      in_tree[i] = false;
      weight[i] = INF;
   }
```

```
    weight[0] = 0;
    parent[0] = -1;

    for(int count = 0; count < MAX_NODES - 1; count++){
        int min_weight = INF, u = -1;

        for(int v = 0; v < MAX_NODES; v++){
            if(!in_tree[v] && weight[v] < min_weight){
                min_weight = weight[v];
                u = v;
            }
        }
        in_tree[u] = true;

        for(int v = 0; v < MAX_NODES; v++){
            if(graph[u][v] && !in_tree[v] && graph[u][v] < weight[v]){
                weight[v] = graph[u][v];
                parent[v] = u;
            }
        }
    }

    printf("\nEDGE    WEIGHT\n");
    for(int i = 1; i < MAX_NODES; i++){
        printf("%d-%d\t%d\n", parent[i], i, weight[i]);
    }
}
```

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

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

```
#define INF 9999
#define MAX_NODES 4

int cost[MAX_NODES][MAX_NODES] = {
   {0, 1, 4, INF},
   {1, 0, 2, 6},
   {4, 2, 0, 3},
   {INF, 6, 3, 0}
};

void distance_routing_algorithm();

int main(){
   printf("Predefined cost matrix\n");
   for(int i = 0; i < MAX_NODES; i++){
      for(int j = 0; j < MAX_NODES; j++){
         if(cost[i][j] == INF){
            printf("INF\t");
         } else {
            printf("%d\t", cost[i][j]);
         }
      }
      printf("\n");
   }

   printf("\nRouting Tables\n");
   distance_routing_algorithm();
   return 0;
}
```

```
void distance_routing_algorithm(){
    int distance[MAX_NODES][MAX_NODES], next_hop[MAX_NODES][MAX_NODES];

    for(int i = 0; i < MAX_NODES; i++){
        for(int j = 0; j < MAX_NODES; j++){
            distance[i][j] = cost[i][j];
            if(cost[i][j] != INF && i != j){
                next_hop[i][j] = j;
            } else {
                next_hop[i][j] = -1;
            }
        }
    }

    for(int k = 0; k < MAX_NODES; k++){
        for(int i = 0; i < MAX_NODES; i++){
            for(int j = 0; j < MAX_NODES; j++){
                if(distance[i][j] > distance[i][k] + distance[k][j]){
                    distance[i][j] = distance[i][k] + distance[k][j];
                    next_hop[i][j] = next_hop[i][k];
                }
            }
        }
    }

    for(int i = 0; i < MAX_NODES; i++){
        printf("Routing tables for node %d\n", i);
        printf("Destination\tCOST\tNEXT_HOP\n");
        for(int j = 0; j < MAX_NODES; j++){
            if(j != i){
                printf("%d\t%d\t", j, distance[i][j]);
```

```
            if(next_hop[i][j] != -1){
                printf("%d\n", next_hop[i][j]);
            } else {
                printf("-\n");
            }
          }
       }
       printf("\n");
   }
}
```

7. Implement data encryption and data decryption

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

void encrypt(char*);
void decrypt(char*);

int main(){
   char text[100];

   printf("Enter the text you want to Encrypt : ");
   fgets(text, sizeof(text), stdin);
   text[strcspn(text, "\n")] = '\0';

   printf("Original Text : %s\n", text);
   encrypt(text);
   printf("Encrypted : %s\n", text);
   decrypt(text);
   printf("Decrypted : %s\n", text);
   return 0;
```

```
}

void encrypt(char* text){
   for(int i = 0; text[i] != '\0'; i++){
      text[i] = text[i] + 1;
   }
}

void decrypt(char* text){
   for(int i = 0; text[i] != 0; i++){
      text[i] = text[i] - 1;
   }
}
```

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

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h> // to use sleep() function in Linux/mac OS

#define MAX_CAPACITY 9
#define LEAK_RATE 2

int bucket_count = 0; // Number of packets currently in the bucket

void arrive_packet(int);
void leak_packets();

int main(){
   int packet_size;

   printf("\nSIMULATION OF LEAKY BUCKET ALGORITHM\n\n");
```

```
    printf("Max bucket capacity : %d\n", MAX_CAPACITY);
    printf("Leak rate of the bucket per second : %d\n\n", LEAK_RATE);

    while(1){
        printf("Enter your packet size (0 to end program) : ");
        scanf("%d", &packet_size);

        if(packet_size == 0){
            break;
        }

        arrive_packet(packet_size);
        sleep(1);
        leak_packets();
    }
    return 0;
}

void arrive_packet(int packet_size){
    if(bucket_count + packet_size <= MAX_CAPACITY){
        bucket_count = bucket_count + packet_size;
        printf("Packet arrived, current bucket size : %d\n", bucket_count);
    } else {
        printf("Packet Discarded, due to OVERFLOW!\n");
    }
}

void leak_packets(){
    if(bucket_count > 0){
        bucket_count = bucket_count - LEAK_RATE;
        if(bucket_count < 0){
```

```
            bucket_count = 0;
        }
        printf("Bucket Leaked! Current Bucket size : %d\n", bucket_count);
    } else {
        printf("Bucket Empty! No packets to Leak!\n");
    }
}
```

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

```
#include<stdio.h>

#define BUFFER_SIZE 10

void insert_frame(int[], int*, int);
void display_buffer(int[], int*);

int main(){
    int count = 0, frame_number, buffer[BUFFER_SIZE];

    while(1){
        printf("Enter your frame number (0 to Quit): ");
        scanf("%d", &frame_number);
        if(frame_number == 0){
            break;
        }
        insert_frame(buffer, &count, frame_number);
        display_buffer(buffer, &count);
    }
    return 0;
}
```

```
void insert_frame(int buffer[], int* count, int frame_number){
    if(*count >= BUFFER_SIZE){
        printf("BUFFER IS FULL, CAN'T INSERT MORE FRAMES\n");
        return;
    }
    int i = (*count) - 1;

    while(i >= 0 && buffer[i] > frame_number){
        buffer[i + 1] = buffer[i];
        i--;
    }
    buffer[i + 1] = frame_number;
    (*count)++;
}

void display_buffer(int buffer[], int* count){
    printf("Current Frames in BUFFER: ");
    for(int i = 0; i < (*count); i++){
        printf("%d ", buffer[i]);
    }
    printf("\n");
}
```