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");

}