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

#include <stdio.h>

#include <string.h>

void char\_count(char \*data) {

printf("\n[Character Count]\nFrame: %d%s\n", (int)strlen(data), data);

}

void char\_stuff(char \*data) {

char res[200]; int i, j = 0;

res[j++] = 'F';

for (i = 0; data[i]; i++) {

if (data[i] == 'F' || data[i] == 'E') res[j++] = 'E';

res[j++] = data[i];

}

res[j++] = 'F'; res[j] = '\0';

printf("\n[Character Stuffing]\nFrame: %s\n", res);

}

void bit\_stuff(char \*data) {

char res[200], flag[] = "01111110";

int i, j = 0, count = 0;

strcpy(res, flag); j = strlen(flag);

for (i = 0; data[i]; i++) {

res[j++] = data[i];

if (data[i] == '1') {

count++;

if (count == 5) { res[j++] = '0'; count = 0; }

} else count = 0;

}

strcat(res, flag);

printf("\n[Bit Stuffing]\nFrame: %s\n", res);

}

int main() {

char data[100];

int choice = 0;

while (choice != 4) {

printf("\n=== Framing Methods ===\n");

printf("1. Character Count\n2. Character Stuffing\n3. Bit Stuffing\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // clear buffer

if (choice == 4) {

printf("\nExiting...\n");

break;

}

printf("Enter data: ");

fgets(data, sizeof(data), stdin);

data[strcspn(data, "\n")] = '\0';

switch (choice) {

case 1: char\_count(data); break;

case 2: char\_stuff(data); break;

case 3: bit\_stuff(data); break;

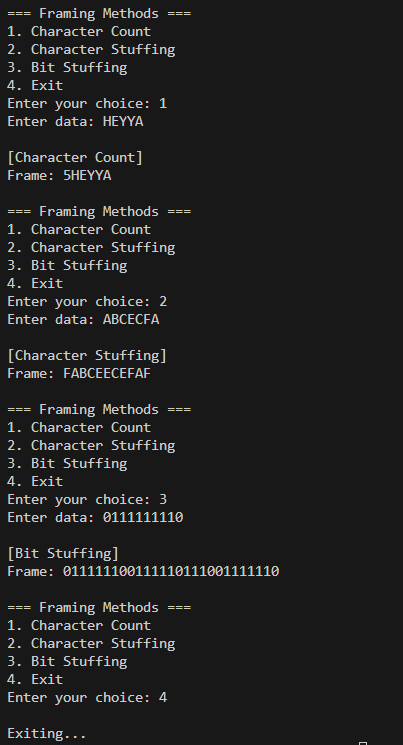
default: printf("\nInvalid choice!\n");

}

}

    return 0;

}

Output: 

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

#include <stdio.h>

#include <string.h>

// Function to perform Modulo-2 division

void computeCRC(char data[], char poly[], char crc[]) {

int dataLen = strlen(data);

int polyLen = strlen(poly);

char temp[256];

strcpy(temp, data);

for (int i = 0; i < polyLen - 1; i++)

strcat(temp, "0");

char remainder[256];

strcpy(remainder, temp);

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

if (remainder[i] == '1') {

for (int j = 0; j < polyLen; j++) {

remainder[i + j] = (remainder[i + j] == poly[j]) ? '0' : '1';

}

}

}

strncpy(crc, &remainder[dataLen], polyLen - 1);

crc[polyLen - 1] = '\0';

}

int main() {

char data[128];

char crc[128], codeword[256];

// Standard Polynomials

char poly12[] = "1100000001111"; // CRC-12

char poly16[] = "11000000000000101"; // CRC-16

char polyCCITT[] = "10001000000100001"; // CRC-CCITT

printf("\n=== CRC Computation for All Polynomials ===\n");

printf("Enter data bits: ");

scanf("%s", data);

printf("\nData bits: %s\n", data);

// --- CRC-12 ---

computeCRC(data, poly12, crc);

strcpy(codeword, data);

strcat(codeword, crc);

printf("\n[CRC-12]");

printf("\nGenerator Polynomial: %s", poly12);

printf("\nCRC bits: %s", crc);

printf("\nTransmitted Codeword: %s\n", codeword);

// --- CRC-16 ---

computeCRC(data, poly16, crc);

strcpy(codeword, data);

strcat(codeword, crc);

printf("\n[CRC-16]");

printf("\nGenerator Polynomial: %s", poly16);

printf("\nCRC bits: %s", crc);

printf("\nTransmitted Codeword: %s\n", codeword);

// --- CRC-CCITT ---

computeCRC(data, polyCCITT, crc);

strcpy(codeword, data);

strcat(codeword, crc);

printf("\n[CRC-CCITT]");

printf("\nGenerator Polynomial: %s", polyCCITT);

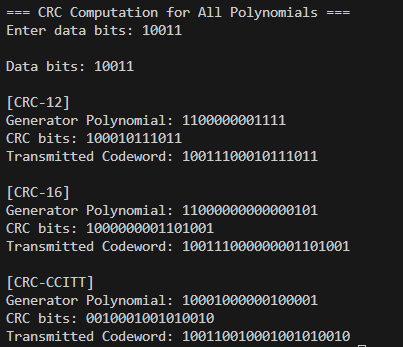
printf("\nCRC bits: %s", crc);

printf("\nTransmitted Codeword: %s\n", codeword);

    return 0;

}

Output:



**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 <stdlib.h>

#include <time.h>

// Generate random percentage

int randPercent() {

return rand() % 100;

}

int main() {

int totalFrames, windowSize, lossChance;

printf("Enter total number of frames to send: ");

scanf("%d", &totalFrames);

printf("Enter window size: ");

scanf("%d", &windowSize);

printf("Enter loss probability (0-100): ");

scanf("%d", &lossChance);

srand(time(0)); // Seed for randomness

int base = 0, next = 0;

printf("\n--- Go-Back-N ARQ Simulation ---\n");

while (base < totalFrames) {

// Send frames in the window

while (next < base + windowSize && next < totalFrames) {

printf("Sender: Frame %d sent.\n", next);

next++;

}

// Simulate a possible frame loss

int lostFrame = (randPercent() < lossChance) ? base + rand() % (next - base) : -1;

if (lostFrame >= 0 && lostFrame < totalFrames) {

printf("Receiver: Frame %d lost!\n", lostFrame);

printf("Sender: Timeout! Go back and resend from Frame %d.\n\n", lostFrame);

next = base = lostFrame; // resend from lost frame

} else {

printf("Receiver: All frames from %d to %d acknowledged.\n\n", base, next - 1);

base = next; // slide window forward

}

}

printf("All frames transmitted successfully!\n");

return 0;

}



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

#include <stdio.h>

#include <limits.h>

int main() {

int n;

printf("Enter number of vertices: ");

scanf("%d", &n);

int graph[20][20];

printf("Enter the adjacency matrix (0 for no edge):\n");

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

scanf("%d", &graph[i][j]);

int src;

printf("Enter source vertex (0 to %d): ", n - 1);

scanf("%d", &src);

int dist[20], visited[20];

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

dist[i] = INT\_MAX;

visited[i] = 0;

}

dist[src] = 0;

// Dijkstra’s Algorithm

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

int u = -1, min = INT\_MAX;

for (int i = 0; i < n; i++)

if (!visited[i] && dist[i] < min) {

min = dist[i];

u = i;

}

if (u == -1) break;

visited[u] = 1;

for (int v = 0; v < n; v++)

if (graph[u][v] && !visited[v] && dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

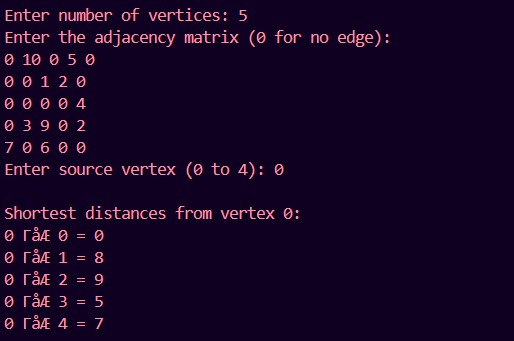
printf("\nShortest distances from vertex %d:\n", src);

for (int i = 0; i < n; i++)

printf("%d → %d = %d\n", src, i, dist[i]);

return 0;

}



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

#include <stdio.h>

int parent[20];

int find(int i) {

while (parent[i] != i)

i = parent[i];

return i;

}

void union\_set(int i, int j) {

int a = find(i);

int b = find(j);

parent[a] = b;

}

int main() {

int cost[20][20];

int n, i, j, edges = 1;

int min, u = -1, v = -1, mincost = 0;

printf("Enter number of hosts (nodes): ");

scanf("%d", &n);

printf("Enter the cost adjacency matrix (use 99 for no connection):\n");

for (i = 1; i <= n; i++) {

for (j = 1; j <= n; j++) {

scanf("%d", &cost[i][j]);

if (cost[i][j] == 0)

cost[i][j] = 99;

}

}

for (i = 1; i <= n; i++)

parent[i] = i;

printf("\nEdges in the Broadcast Tree:\n");

while (edges < n) {

min = 99;

for (i = 1; i <= n; i++) {

for (j = 1; j <= n; j++) {

if (find(i) != find(j) && cost[i][j] < min) {

min = cost[i][j];

u = i;

v = j;

}

}

}

union\_set(u, v);

printf("%d -> %d cost = %d\n", u, v, min);

mincost += min;

cost[u][v] = cost[v][u] = 99;

edges++;

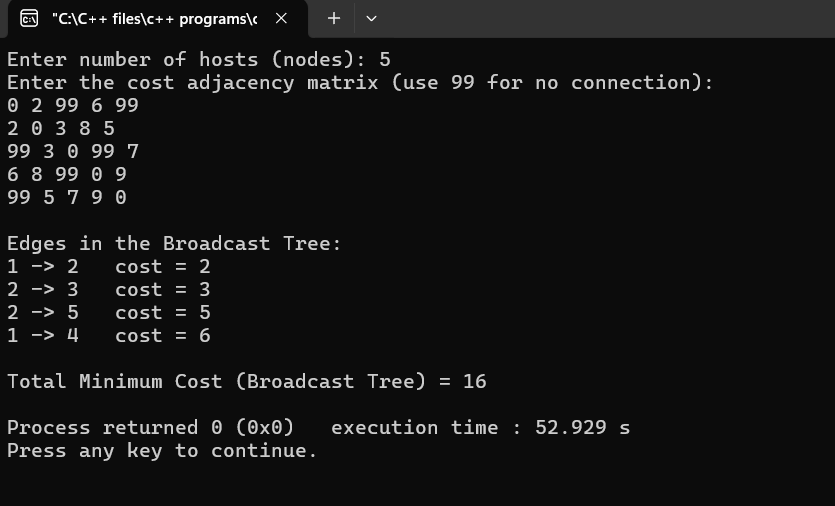
}

printf("\nTotal Minimum Cost (Broadcast Tree) = %d\n", mincost);

return 0;

}

**Output:**

****

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

**Program:**

#include <stdio.h>

struct node {

int dist[10]; // Distance to other nodes

int from[10]; // From which node the packet will come

} rt[10];

int main() {

int costmat[10][10];

int nodes, i, j, k, count = 0;

printf("Enter the number of nodes: ");

scanf("%d", &nodes);

printf("Enter the cost matrix (999 for infinity):\n");

for (i = 0; i < nodes; i++) {

for (j = 0; j < nodes; j++) {

scanf("%d", &costmat[i][j]);

costmat[i][i] = 0;

rt[i].dist[j] = costmat[i][j];

rt[i].from[j] = j;

}

}

do {

count = 0;

for (i = 0; i < nodes; i++) {

for (j = 0; j < nodes; j++) {

for (k = 0; k < nodes; k++) {

if (rt[i].dist[j] > costmat[i][k] + rt[k].dist[j]) {

rt[i].dist[j] = costmat[i][k] + rt[k].dist[j];

rt[i].from[j] = k;

count++;

}

}

}

}

} while (count != 0);

printf("\nFinal Routing Tables:\n");

for (i = 0; i < nodes; i++) {

printf("\nRouter %d Table:\n", i + 1);

printf("Destination\tNext Hop\tDistance\n");

for (j = 0; j < nodes; j++) {

printf("%d\t\t%d\t\t%d\n", j + 1, rt[i].from[j] + 1, rt[i].dist[j]);

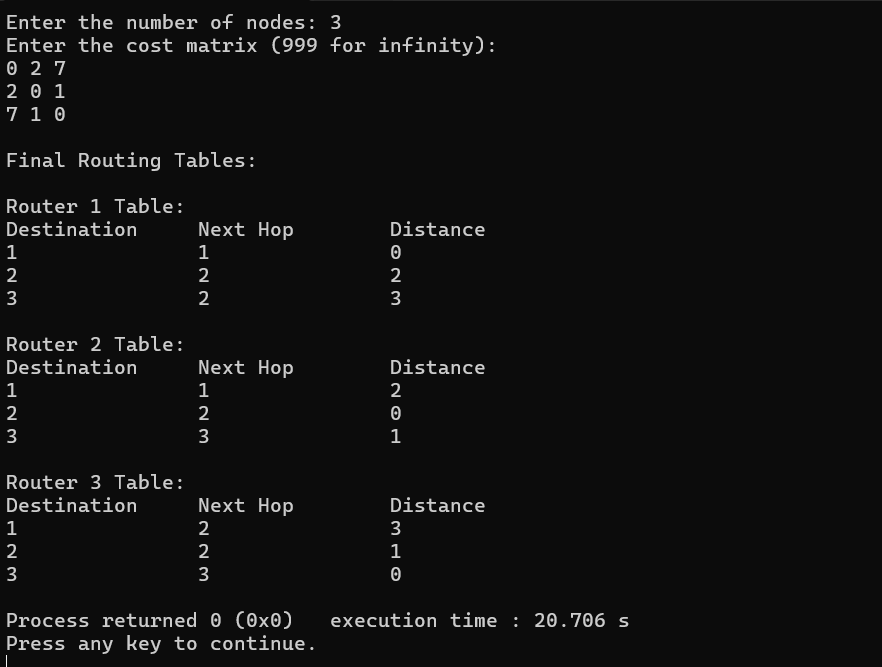
}

}

return 0;

}

**Output:**



**7. Implement data encryption and data decryption**

#include <stdio.h>

#include <string.h>

void encrypt(char \*text, int key) {

for (int i = 0; text[i] != '\0'; i++) {

if (text[i] >= 'A' && text[i] <= 'Z') {

text[i] = 'A' + (text[i] - 'A' + key) % 26;

} else if (text[i] >= 'a' && text[i] <= 'z') {

text[i] = 'a' + (text[i] - 'a' + key) % 26;

}

}

}

void decrypt(char \*text, int key) {

for (int i = 0; text[i] != '\0'; i++) {

if (text[i] >= 'A' && text[i] <= 'Z') {

text[i] = 'A' + (text[i] - 'A' - key + 26) % 26;

} else if (text[i] >= 'a' && text[i] <= 'z') {

text[i] = 'a' + (text[i] - 'a' - key + 26) % 26;

}

}

}

int main() {

char text[100];

int key;

printf("Enter text: ");

fgets(text, sizeof(text), stdin);

text[strcspn(text, "\n")] = '\0';

printf("Enter key (number): ");

scanf("%d", &key);

printf("\nOriginal text: %s\n", text);

encrypt(text, key);

printf("Encrypted text: %s\n", text);

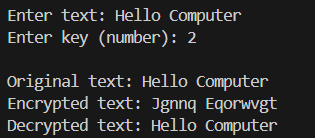
decrypt(text, key);

printf("Decrypted text: %s\n", text);

    return 0;

}

Output:



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

#include <stdio.h>

int main() {

int capacity, leakRate, current = 0;

int incoming, time = 0;

printf("Enter bucket capacity: ");

scanf("%d", &capacity);

printf("Enter leak rate: ");

scanf("%d", &leakRate);

while (1) {

time++;

printf("\nTime %d - Enter incoming packets (-1 to stop): ", time);

scanf("%d", &incoming);

if (incoming == -1)

break;

// Add incoming packets

if (current + incoming > capacity) {

int dropped = (current + incoming) - capacity;

current = capacity;

printf("Bucket overflow! %d packets dropped.\n", dropped);

} else {

current += incoming;

printf("%d packets added.\n", incoming);

}

// Leak packets

if (current < leakRate)

current = 0;

else

current -= leakRate;

printf("%d packets leaked. Packets remaining: %d\n", leakRate, current);

}

printf("\nSimulation ended.\n");

return 0;

}

Output:

