CHARCOUNT

```
#include<stdio.h>
#include<string.h>
int main()
{
int n,i;
printf("enter no.of frames:");
scanf("%d",&n);
char frames[n][50];
for (int i=0;i<n;i++)
{
printf("enter frame %d",i);
scanf("%s",frames[i]);
}
printf("\n char count frames data:\n");
for(int i=0;i<n;i++)
{
int length=strlen(frames[i])+1;
printf("%d%s",length,frames[i]);
}
printf("\n");
return 0;
}
```

BIT STUFFING

```
#include <stdio.h>
#include <string.h>
int main() {
    int a[20], b[30];
    int i, j = 0, k, count = 1, n;
    printf("Enter frame size: ");
    scanf("%d", &n);
    printf("Enter the frame (0s and 1s):\n");
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    i = 0;
    b[j] = a[i];  // first bit copied
    j++;
    for (i = 1; i < n; i++) {
        b[j] = a[i];
        j++;
        if (a[i] == 1) {
            count++;
        } else {
            count = 0;
        }
        if (count == 5) { // if 5 ones in a row, stuff a 0
            b[j] = 0;
            j++;
            count = 0; }
    } printf("After bit stuffing, the frame is:\n");
    for (i = 0; i < j; i++)
 printf("%d", b[i]);
printf("\n");
```

```
    return 0;
}
```

CHARACTER STUFFING

```
#include <stdio.h>
#include <string.h>
int main() {
    char data[50], stuffed[100];
    char start, end;
    int i, j = 0;
    printf("Enter the data to be stuffed: ");
    scanf("%s", data);
    printf("Enter the starting delimiter character: ");
    scanf(" %c", &start);
    printf("Enter the ending delimiter character: ");
    scanf(" %c", &end);
    stuffed[j++] = start;
    stuffed[j++] = start;  /
    for (i = 0; i < strlen(data); i++) {
        stuffed[j++] = data[i];
        if (data[i] == start || data[i] == end) {
            stuffed[j++] = data[i];  } }
   stuffed[j++] = end;
    stuffed[j++] = end;
    stuffed[j] = '\0';
    printf("Stuffed data: %s\n", stuffed);
 return 0;}
```

BROADCASTTREE

```
#include <stdio.h>
#define max 10
```

```
int n, a[max][max];
int visited[max];
void broadcasttree(int root) {
    int queue[max], front = 0, rear = 0;
    for (int i = 0; i < n; i++)
        visited[i] = 0;
    visited[root] = 1;
    queue[rear++] = root;
    printf("\nBroadcast tree edges:\n");
    while (front < rear) {
        int current = queue[front++];
        for (int j = 0; j < n; j++) {
            if (a[current][j] == 1 && visited[j] == 0) {
                visited[j] = 1;
                queue[rear++] = j;
                printf("Host%d => Host%d\n", current + 1, j + 1);
            }
        }
        }
}
int main() {
    int i, j, root;
    printf("Enter no. of nodes: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf(" enter Connection of %d => %d : ", i + 1, j + 1);
            scanf("%d", &a[i][j]);
```

```
        }
    }
    printf("Enter root node: ");
    scanf("%d", &root);
    broadcasttree(root - 1);
    return 0;
}
```

VECTOR ROUTING ALGORITHM

```
#include <stdio.h>
#define INF 9999
int main() {
    int nv, sn, nodej;
    int edel[20], tdel[20][20], min;
    char sv, adver[20], ch;
    printf("Enter the number of vertices: ");
    scanf("%d", &nv);
    printf("Enter the source vertex number and name: ");
    scanf("%d %c", &sn, &sv);
    printf("Enter the number of adjacent vertices to vertex %c: ", sv);
    scanf("%d", &nodej);
    for (int i = 0; i < nodej; i++) {
        printf("Enter time delay and name of adjacent vertex %d: ", i + 1);
        scanf("%d %c", &edel[i], &adver[i]); // Added space before %c
    }
    for (int i = 0; i < nodej; i++) {
        printf("Enter the time delay from %c to all %d vertices:\n", adver[i], nv);
        for (int j = 0; j < nv; j++) {
```

```
            scanf("%d", &tdel[i][j]);
        }
    }
    printf("\nDelay\tVia Vertex\n");
    for (int i = 0; i < nv; i++) {
        if (i == sn - 1) {
            printf("0\t-\n");
            continue;
        }
      min = INF;
        ch = '-';
        for (int j = 0; j < nodej; j++) {
            int total_delay = tdel[j][i] + edel[j];
            if (total_delay < min) {
                min = total_delay;
                ch = adver[j];
            }
        }
       printf("%d\t%c\n", min, ch);
    }
    return 0;
}
```

ENCRYPTION DECRYPTION

```
#include<stdio.h>
#include<string.h>
void Xorcipher(char *data,char key){
for(int i=0; data[i]!='\0';i++){
```

```
data[i]=data[i]^key;
}}
int main(){
char text[100];
char key;
printf("enter the text:");
scanf("%[^\n]s",text);
getchar();
printf("enter a single character as key:");
scanf("%c",&key);
getchar();
Xorcipher(text,key);
printf("encrypted text(in flex):");
for(int i=0;text[i]!='\0';i++){
printf("%02x",(unsigned char)text[i]);
}
Xorcipher(text,key);
printf("\n decrypted text:%s\n",text);
return 0;
}
```

FRAME SORTING

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int frame_no;
    char data[50];
```

```
}
frame;
int compare(const void *a, const void *b) {
    return ((frame *)a)->frame_no - ((frame *)b)->frame_no;
}
int main()
{
    int n;
    printf("Enter the number of frames received: ");
    scanf("%d", &n);
    frame buffer[n];
    printf("Enter the frame number and data count order:\n");
    for (int i = 0; i < n; i++) {
        printf("\nFrame %d:\n", i + 1);
        printf("Frame number: ");
        scanf("%d", &buffer[i].frame_no);
        printf("Data: ");
        scanf("%s", &buffer[i].data);
    }
    qsort(buffer, n, sizeof(frame), compare);
    printf("\nSorted frames:\n");
    for (int i = 0; i < n; i++) {
        printf("Frame %d: %s\n", buffer[i].frame_no, buffer[i].data);
    }
return 0;
}
```

SLIDING WINDOW PROTOCOL

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

```
#include <stdlib.h>
#include <time.h>
#define TOTAL_FRAMES 10
#define WINDOW_SIZE 4
#define LOSS_PROBABILITY 20   // chance (in %) of a frame being lost
int is_frame_lost() {
    return (rand() / 100) < LOSS_PROBABILITY;  // returns 1 (true) if frame lost
}
int main() {
    int base = 0;          // starting frame of the current window
    int next_seq_num = 0;  // next frame number to send
    int i;
    srand(time(NULL));
    printf("Simulating Go-Back-N ARQ with Sliding Window Protocol\n\n");
    while (base < TOTAL_FRAMES) {
        printf("Sender window: [%d to %d]\n", base, base + WINDOW_SIZE - 1);
        for (i = base; i < base + WINDOW_SIZE && i < TOTAL_FRAMES; i++) {
            printf("Sending frame %d... ", i);
            if (is_frame_lost()) {
                printf("[Lost]\n");
                break;  // simulate loss — stop sending and trigger retransmission
            } else {
                printf("[Received by Receiver]\n");
            }
        }
        if (i == base + WINDOW_SIZE || i >= TOTAL_FRAMES) {
            base += WINDOW_SIZE;
            if (base > TOTAL_FRAMES)
                base = TOTAL_FRAMES;
```

```
            printf("Receiver ACKs up to frame %d\n\n", base - 1);
        } else {
            // Frame loss happened — Go Back N retransmission
            printf("Receiver did NOT ACK for frame %d, retransmitting window...\n\n", i);
        }
    }
    printf("All frames sent successfully using Go-Back-N ARQ!\n");
    return 0;
}
```

LEAKY BUCKET

```
#include <stdio.h>
#include <stdlib.h>
#define BUCKET_SIZE 10
void leakybucket(int incoming[], int n, int output_rate) {
    int i, bucket_content = 0;
    printf("Time\tIncoming\tBucketContent\tOutgoing\tRemaining\n");
    for (i = 0; i < n; i++) {
        printf("%d\t",i+1);
        printf("%d\t\t", incoming[i]);
        if (incoming[i] + bucket_content > BUCKET_SIZE) {
            printf("Bucket Overflow!\n");
            bucket_content = BUCKET_SIZE;
        } else {
            bucket_content += incoming[i];
            printf("%d\t\t", bucket_content);
        }
        int sent = (bucket_content < output_rate) ? bucket_content : output_rate;
        bucket_content -= sent;
```

```
        printf("%d\t\t%d\n", sent, bucket_content);
    }
    while (bucket_content > 0) {
        int sent = (bucket_content < output_rate) ? bucket_content : output_rate;
        printf("Extra\t-\t\t%d\t\t%d\t\t", bucket_content, sent);
        bucket_content -= sent;
        printf("%d\n", bucket_content);
    }
}
int main() {
    int n, output_rate;
    printf("Enter number of time intervals: ");
    scanf("%d", &n);
    int incoming[n];
    printf("Enter incoming packets for each interval:\n");
    for (int i = 0; i < n; i++) {
        printf("Interval %d: ", i + 1);
        scanf("%d", &incoming[i]);
    }
    printf("Enter output rate: ");
    scanf("%d", &output_rate);
    leakybucket(incoming, n, output_rate);
    return 0;
}
```