Bit Stuffing

```
#include <stdio.h>
#define MAX 100

int main() {
   int input[MAX], stuffed[MAX * 2], unstuffed[MAX], count = 0, n, stuffedIndex = 0, unstuffedIndex 0;

   printf("Enter the number of bits: ");
   scanf("%d", &n);
   printf("Enter the bits (0s and 1s): ");
   for (int i = 0; i < n; i++) {
      scanf("%d", &input[i]);
   }

   for (int i = 0; i < n; i++) {
      stuffed[stuffedIndex++] = input[i];
      if (input[i] == 1) {
         count++;
         if (count == 5) {
            stuffed[stuffedIndex++] = 0;
            count = 0;
         }
      } else {
         count = 0;
      }
   }

   printf("Stuffed data: ");
   for (int i = 0; i < stuffedIndex; i++) {
      printf("%d", stuffed[i]);
   }
```

```
    printf("\n");

    count = 0;
    for (int i = 0; i < stuffedIndex; i++) {
        unstuffed[unstuffedIndex++] = stuffed[i];
        if (stuffed[i] == 1) {
            count++;
            if (count == 5) {
                i++;
                count = 0;
            }
        } else {
            count = 0;
        }
    }

    printf("Unstuffed data: ");
    for (int i = 0; i < n; i++) {
        printf("%d", unstuffed[i]);
    }
    printf("\n");

    return 0;
}
```

Char Stuffing

```
#include <stdio.h>

int main() {
    char data[50], stuffed[50], unstuffed[50];
    char flag = 'f', esc = 'e';
    int n, i, stuffedIndex = 0, unstuffedIndex = 0;
```

```
printf("Enter the length of data: ");
scanf("%d", &n);
printf("Enter the characters or data: ");

for (i = 0; i < n; i++) {
   scanf(" %c", &data[i]);
}

stuffed[stuffedIndex++] = flag;

for (i = 0; i < n; i++) {
   if (data[i] == flag || data[i] == esc) {
      stuffed[stuffedIndex++] = esc;
   }
   stuffed[stuffedIndex++] = data[i];
}

stuffed[stuffedIndex++] = flag;

printf("Stuffed data: ");
for (i = 0; i < stuffedIndex; i++) {
   printf("%c", stuffed[i]);
}
printf("\n");

for (i = 1; i < stuffedIndex - 1; i++) {
   if (stuffed[i] == esc) {
      i++;
   }
   unstuffed[unstuffedIndex++] = stuffed[i];
```

```
    }

    printf("Unstuffed data: ");
    for (i = 0; i < unstuffedIndex; i++) {
        printf("%c", unstuffed[i]);
    }
    printf("\n");

    return 0;
}
```

CRC

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

int main() {
    char message[200], generator[50], transmitted[200], received[200];
    int msgLen, genLen, i, j;

    // Input the message and generator
    printf("Enter the message: ");
    scanf("%s", message);
    printf("Enter the generator polynomial: ");
    scanf("%s", generator);

    msgLen = strlen(message);
    genLen = strlen(generator);

    // Create dividend by appending zeros
    char dividend[200];
```

```
strcpy(dividend, message);
for (i = msgLen; i < msgLen + genLen - 1; i++) {
   dividend[i] = '0';
}
dividend[msgLen + genLen - 1] = '\0';
printf("Dividend is: %s\n",dividend);
// Perform division to calculate CRC
for (i = 0; i <= msgLen; i++) {
   if (dividend[i] == '1') {
      for (j = 0; j < genLen; j++) {
         // if (dividend[i + j] == generator[j]) {
         //     dividend[i + j] = '0';
         // } else {
         //     dividend[i + j] = '1';
         // }
         dividend[i+j] = (dividend[i+j] == generator[j])? '0':'1';          }
   }
}

// Extract and print CRC
printf("Calculated CRC: %s\n", dividend + msgLen);

// Append CRC and display the transmitted message
strcpy(transmitted, message);
strcat(transmitted, dividend + msgLen);
printf("Transmitted message: %s\n", transmitted);

// Check received message for errors
```

```
    printf("Enter the received message: ");
    scanf("%s", received);
    strcpy(dividend, received);
    for (i = 0; i <= strlen(received) - genLen; i++) {
        if (dividend[i] == '1') {
            for (j = 0; j < genLen; j++)
            {
                // Perform bitwise subtraction (XOR logic without XOR)
                if (dividend[i + j] == generator[j]) {
                    dividend[i + j] = '0';
                } else {
                    dividend[i + j] = '1';
                }
            }
        }
    }

    // Check for errors by verifying if the remainder is all zeros
    if (strspn(dividend + strlen(received) - genLen + 1, "0") == genLen - 1)
        printf("The received message is correct.\n");
    else
        printf("The received message has errors.\n");

    return 0;

}
```

IP FRAG

#include <stdio.h> #include <stdlib.h>

void fragmentIPDatagram(int packetSeqNum, int datagramSize, int mtu, int DF) { int headerSize = 20;

int fragmentDataSize = mtu- headerSize; int totalData = datagramSize - headerSize;

int numFragments = (totalData + fragmentDataSize- 1) / fragmentDataSize; int offset = 0;

printf("\nPacket Sequence Number: %d\n", packetSeqNum);

printf("Frag No.\tDF\tMF\tFragment Offset\tFragment Size\n");

if (DF && datagramSize > mtu) {

printf("Fragmentation not allowed (DF = 1). Packet too large for MTU.\n"); return;

}

for (int i = 1; i <= numFragments; i++) {

```
if (totalData > fragmentDataSize) {
currentDataSize = fragmentDataSize;
} else {
currentDataSize = totalData;

}
printf("%d\t\t%d\t%d\t%d\t\t%d\n", i,
DF,
(i == numFragments) ? 0 : 1; offset,
currentDataSize + headerSize
);



totalData-= currentDataSize; offset += currentDataSize / 8;
}
}



int main() {
int packetSeqNum, datagramSize, mtu, DF;



printf("Enter the packet sequence number: ");
```

```
printf("Enter the total size of the IP datagram (in bytes): "); scanf("%d",
&datagramSize);

printf("Enter the Maximum Transmission Unit (MTU) (in bytes): "); scanf("%d",
&mtu);

printf("Enter the Don't Fragment (DF) flag (0 or 1): "); scanf("%d", &DF);




if (datagramSize <= 20) {
printf("Error: Datagram size must be greater than the header size (20 bytes).\n");
return 1;
}
if (mtu <= 20) {
printf("Error: MTU must be greater than the header size (20 bytes).\n"); return 1;
}
fragmentIPDatagram(packetSeqNum, datagramSize, mtu, DF);

return 0;
}
```

DVR

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

```
#define MAX 50
#define INF 9999
int main()
{
int n, i, j, k;
char router[MAX][MAX];int dist[MAX][MAX], nexthop[MAX][MAX];
printf("Enter the number of nodes: "); scanf("%d", &n);
printf("Enter the distances between nodes (use-1 for no path):\n"); for(i = 0; i < n;
i++) {
for(j = 0; j < n; j++) { if(i == j) {
dist[i][j] = 0; continue;
}
printf("dist[%d][%d]: ", i, j);
scanf("%d", &dist[i][j]);
if(dist[i][j] ==-1) { dist[i][j] = INF;
}
}
}
printf("Enter the names of the nodes:\n"); for(i = 0; i < n; i++) {
scanf("%s", router[i]);
}
```

```
for(i = 0; i < n; i++) { for(j = 0; j < n; j++) {
nexthop[i][j] = (dist[i][j] != INF && i != j) ? j :-1;
}
}
for(k = 0; k < n; k++) { for(i = 0; i < n; i++) {
for(j = 0; j < n; j++) {
if(dist[i][k] != INF && dist[k][j] != INF && dist[i][j] > dist[i][k] + dist[k][j])
{
dist[i][j] = dist[i][k] + dist[k][j]; nexthop[i][j] = nexthop[i][k];
}
}
}
}
for(i = 0; i < n; i++) {
printf("\nRouting table for node: %s\n", router[i]);
printf("Destination\tDistance\tNextHop\n");
for(j = 0; j < n; j++) { if(i == j) {
continue;
}
if(dist[i][j] == INF) {
```

```
} else {
printf("%s\t\t%d\t\t%s\n", router[j], dist[i][j], router[nexthop[i][j]]);
}
}
}
return 0;
}
```

Shortest Path

```
#include<stdio.h>
 #include<stdlib.h>
  #define MAX 50
#define INF 9999
int main() {
int n, src, dest, i, j;
int dist[MAX][MAX], cost[MAX], visited[MAX], parent[MAX];
printf("Enter the number of nodes: "); scanf("%d", &n);
printf("Enter the distance matrix (if no connection, enter-1):\n"); for(i = 0; i < n; i++) {
for(j = 0; j < n; j++) { if(i == j) {
dist[i][j] = 0; continue;
}
```

```
dist[i][j] = dist[j][i]; continue;
}
printf("dist[%d][%d]: ", i, j);
scanf("%d", &dist[i][j]);
if(dist[i][j] ==-1) { dist[i][j] = INF;
}
}
}
printf("\nEnter the source node (0 to %d): ", n- 1); scanf("%d", &src);
printf("Enter the destination node (0 to %d): ", n- 1); scanf("%d", &dest);
for(i = 0; i < n; i++) { cost[i] = INF; visited[i] = 0;
parent[i] =-1;
}
cost[src] = 0;
for(i = 0; i < n; i++) {
int min = INF, min_index =-1;
for(j = 0; j < n; j++) {
if(!visited[j] && cost[j] < min) { min = cost[j];
min_index = j;
}
}
if(min_index ==-1) { break;
}
visited[min_index] = 1;
for(j = 0; j < n; j++) {
if(!visited[j] && dist[min_index][j] != INF) { if(cost[min_index] + dist[min_index][j] <
cost[j]) {
cost[j] = cost[min_index] + dist[min_index][j]; parent[j] = min_index;
```

```
}
}
}
}
```

```
int path[MAX], p = 0;
int temp = dest;
printf("\nThe shortest path: "); if(cost[dest] == INF) {
printf("No path exists between source and destination.\n");
} else {
while(temp !=-1) {
path[p++] = temp; temp = parent[temp];
}
for(i = p- 1; i >= 0; i--) {
printf("%d ", path[i]);
}
printf("\nTotal cost: %d\n", cost[dest]);
}
return 0;
}
```

Go Back N

```
#include<stdio.h>
int main(){
   int window_size, sent = 0, ack, i;
   printf("Enter the window size: ");
```

```
    scanf("%d", &window_size);
    while(1){
        for(int i = 1; i<=window_size; i++){
            printf("Frame %d has been succesfully transmitted \n", sent);
            sent ++;
            if (sent == window_size)
            break;
        }
        printf("Enter the last acknowledgement recieved: ");
        scanf("%d", &ack);
        if (ack == window_size)
        break;
        else
        sent = ack;
    }
    return 0;
}
```

Token Bucket

```
#include <stdio.h>

int main() {
  int tokens = 0, tokenrate, bucketsize, packetsize, time = 0;

  // Get the bucket size from the user
  printf("Enter the bucket size: ");
  scanf("%d", &bucketsize);

  // Get the token rate from the user
```

```
printf("Enter the token rate: ");
scanf("%d", &tokenrate);

// Prompt user to enter the packet size for each time interval
printf("Enter the packet size for each time interval (-1 to stop):\n");

while (1) {
 // Accumulate tokens in the bucket up to the bucket size
 if (tokens + tokenrate <= bucketsize) {
  tokens += tokenrate;
 } else {
  tokens = bucketsize;
 }

 // Get the packet size for the current time interval
 printf("For time %d: ", time);
 scanf("%d", &packetsize);

 // Stop if the packet size is -1
 if (packetsize == -1) {
  break;
 }

 // Check for invalid packet size
 if (packetsize < 0) {
  printf("Invalid packet size\n");
  continue;
 }
```

```
    // Send packet if there are enough tokens
    if (packetsize <= tokens) {
      tokens -= packetsize;
      printf("Packet sent\n");
    } else {
      printf("Packet not sent due to insufficient tokens\n");
    }

    // Display remaining tokens
    printf("Remaining tokens: %d\n", tokens);
    time++;
  }

  return 0;
}
```

Leaky Bucket

```
#include <stdio.h>

void leakybucket(int bucketsize, int outgoingrate) {
  int time = 0, buffer = 0, packetsize;
  printf("Enter the packet size for each time interval (-1 to stop):\n");
  while (1) {
    printf("For time %d: ", time);
    scanf("%d", &packetsize);
    if (packetsize == -1) {
      break;
    }
```

```
    if (packetsize <= (bucketsize - buffer)) {
      buffer += packetsize;
      printf("Packet accepted\n");
    } else {
      printf("Packet rejected\n");
    }
    if (buffer > outgoingrate) {
      buffer -= outgoingrate;
    } else {
      buffer = 0;
    }
    printf("Current buffer size is %d\n", buffer);
    time++;
  }
}

int main() {
  int bucketsize, outgoingrate;
  printf("Enter bucket size: ");
  scanf("%d", &bucketsize);
  printf("Enter outgoing rate: ");
  scanf("%d", &outgoingrate);
  leakybucket(bucketsize, outgoingrate);
  return 0;
}
```