# Computer Networks Lab

1. Write a program for error detecting code using 16 bits CRC-CCITT (Consultative Committee for International Telephony and Telegraphy).

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

# define MAX 30

/* crc(dividend, divisor, remainder) */
void crc(char *data, char *gen, char *rem)
{
    int i, j, k=0;
    char out[MAX]; // xored val after each step

    strcpy(out, data);

    /* Perform XOR on the msg */
    for(i=0; i<strlen(data)-strlen(gen)+1; i++)
    {
        if(out[i] == '1')
        {
            out[i] = '0' ;
            for(j=1; j<strlen(gen); j++)
            {
                out[i+j] = (out[i+j] == gen[j]) ? '0' : '1';
            }
        }
    }

    int idx = strlen(out)-strlen(gen)+1;
    for(i=0; i<strlen(gen)-1; i++)
    {
        rem[i] = out[idx+i];
    }
}

int main()
{
    int i, j;

    char dword[MAX]; // dataword
    char augWord[MAX]; // augmented dataword
    char cword[MAX]; // codeword
    char rem[MAX]; // remainder from crc
    char recv[MAX]; // received message
    char gen[] = "10001000000100001";

    printf("\nCRC-16 Generator : x^16 + x^12 + x^5 + 1 ");
    printf("\nBinary Form      : %s", gen);
    printf("\n\nEnter Dataword   : ");
    scanf("%s", dword);

    strcpy(augWord, dword);
    for(i=0; i<strlen(gen)-1; i++)
    {
        strcat(augWord, "0");
    }
    printf("\nAugmented dataword is   : %s",augWord);

    crc(augWord, gen, rem);

    strcpy(cword, dword);
    strcat(cword, rem);
    printf("\n\nFinal data transmitted  : %s", cword);

    printf("\n\nEnter the data received : ");
    scanf("%s", recv);
    if(strlen(recv) < strlen(cword))
    {
        printf("\n Invalid input \n");
        exit(0);
    }

    crc(recv, gen, rem);

    printf("\nSyndrome = %s ", rem);
    for(i=0; i<strlen(rem); i++)
    {
        if(rem[i] == '1')
        {
            printf("\nError occured !!! Corrupted data received. \n");
            exit(0);
        }
    }
    printf("\nNo Error. Data received successfully.\n");
}

/*************** Output -1 *************************
CRC-16 Generator : x^16 + x^12 + x^5 + 1
Binary Form      : 10001000000100001

Enter Dataword   : 11110001

Augmented dataword is   : 111100010000000000000000

Final data transmitted  : 111100011111111100111110

Enter the data received : 111100011111111100111110

Syndrome = 0000000000000000
No Error. Data received successfully.
******************************************************/
```

```
/*************** Output -2 ************************
CRC-16 Generator : x^16 + x^12 + x^5 + 1
Binary Form      : 10001000000100001

Enter Dataword   : 10101011

Augmented dataword is   : 1.0101011e+23

Final data transmitted  : 101010110000010010000001

Enter the data received : 101010110000000000000000

Syndrome = 0000010010000001
Error occured !!! Corrupted data received.
****************************************************/
```

2. Write a program to divide the message into variable length frames and sort them and display the message at the receiving side.

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

#define MAX 100

typedef struct{
    int id;
    char data[MAX];
}frame;

// Fisher yates algorithm to shuffle the frame
void shuffleFrame(frame f[MAX], int n)
{
    srand(time(NULL));

    int i;
    for(i=n; i>=0; i--)
    {
        int j = rand()%(i+1);

        frame temp = f[j];
        f[j] = f[i];
        f[i] = temp;
    }
}

// Insertion sort algorithm to sort frames based on id
void sortFrames(frame f[MAX], int n)
{
    int i, j;
```

```c
    for(i=1; i<=n; i++)
    {
        frame t = f[i];
        j = i-1;
        while(j>=0 && f[j].id > t.id)
        {
            f[j+1] = f[j];
            j=j-1;
        }
        f[j+1] = t;
    }
}

int main()
{
    frame f[MAX];
    int n;    // no of frames
    int fsize; // size of frame

    char msg[MAX];
    int m = 0; // message iterator
    int i, j;

    printf("Enter a message : ");
    fgets(msg , MAX, stdin);
    msg[strlen(msg)-1] = '\0'; // to remove '\n' from string

    printf("Enter size of the frame : ");
    scanf("%d" , &fsize);

    n = strlen(msg) / fsize ;   // find the number of frames

    // Divide the message into frames
    for(i=0 ; msg[i] != '\0' ; i++)
    {
        f[i].id = i;

        for(j=0 ; j<fsize && m <= strlen(msg); j++)
        {
            f[i].data[j] = msg[m++];
        }

    }

    shuffleFrame(f, n);

    printf("\nShuffled frames:");
    printf("\nframe_id \t frame_data \n");
    printf("----------------------------\n");
    for(i=0 ; i <= n; i++)
    {
        printf("%d \t\t %s \n", f[i].id, f[i].data);
    }
    sortFrames(f, n);

    printf("\nSorted frames:");
    printf("\nframe_id \t frame_data \n");
    printf("----------------------------\n");
    for(i=0 ; i <= n; i++)
    {
        printf("%d \t\t %s \n", f[i].id, f[i].data);
    }

    printf("\nfinal message : ");
    for(i=0; i<= n; i++)
    {
        printf("%s", f[i].data);
    }

    printf("\n");
}


/**************** OUTPUT-1 ************************
Enter a message : network programming lab
Enter size of the frame : 4

Shuffled frames:
frame_id         frame_data
----------------------------
5                lab
1                ork
2                prog
0                netw
3                ramm
4                ing

Sorted frames:
frame_id         frame_data
----------------------------
0                netw
1                ork
2                prog
3                ramm
4                ing
5                lab

final message : network programming lab
*************************************************/
```

3. For the given network graph, write a program to implement Link state routing algorithm to build a routing table for the given node.

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

```c
#define INFINITY 999
#define MAX 100

int cost[MAX][MAX]; // cost matrix
int distance[MAX]; // distance from source
int visited[MAX] = {0};
int parent[MAX];
int source;
int n; // number of nodes

void initialize()
{
    int i;
    visited[source] = 1;
    parent[source] = source;

    for(i=0; i<n; i++)
    {
        distance[i] = cost[source][i];
        if( cost[source][i] != INFINITY )
        {
            parent[i] = source;
        }
    }
}

/* Get Minimum Node Not In Network */
int GetMin()
{
    int minIdx = -1;
    int minDist = INFINITY;

    int i;
    for(i=0; i<n; i++)
    {
        if( !visited[i] && minDist >= distance[i] )
        {
                minIdx = i;
                minDist = distance[i];
        }
    }
    return minIdx;
}

/* update distance for adjacent nodes */
void updateTable(int node)
{
    int i;
    for(i=0; i<n; i++)
    {
        if( cost[node][i] != INFINITY && distance[i] > distance[node]+cost[node][i] )
        {
            distance[i] = distance[node] + cost[node][i];
            parent[i] = node;
        }
    }
}

void display()
{
    int i;
    int node;

    printf("\nNode \t Distance from source \t Path \n");
    for(i=0; i<n; i++)
    {
        printf("%d \t\t %d \t\t", i, distance[i]);

        // node <- parent[node] <- parent[parent[node]] <- ... <- source
        node = i;
        printf("%d", node);
        while( node != source)
        {
            printf(" <- %d", parent[node]);
            node = parent[node];
        }
        printf("\n");
    }
}

int main()
{
    int i, j, node;

    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    printf("Enter the source node    : ");
    scanf("%d", &source);

    printf("\nEnter the cost matrix: \n");
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            scanf("%d", &cost[i][j]);
        }
    }

    initialize();

    for(i=0; i<n-1; i++) // for all remaining vertices(since source is already visited)
    {
        node = GetMin();
        visited[node] = 1;
        updateTable(node);
    }
    display();
    return 0;
}


/*************** OUTPUT-1 ******************************
Enter the number of nodes: 9
Enter the source node    : 3

Enter the cost matrix:
0     4     999   999   999   999   999   8     999
4     0     8     999   999   999   999   11    999
999   8     0     7     999   4     999   999   2
999   999   7     0     9     14    999   999   999
999   999   999   9     0     10    999   999   999
999   999   4     14    10    0     2     999   999
999   999   999   999   999   2     0     1     6
8     11    999   999   999   999   1     0     7
999   999   2     999   999   999   6     7     0

Node     Distance from source    Path
0                19             0 <- 1 <- 2 <- 3
1                15             1 <- 2 <- 3
2                7              2 <- 3
3                0              3
4                9              4 <- 3
5                11             5 <- 2 <- 3
6                13             6 <- 5 <- 2 <- 3
7                14             7 <- 6 <- 5 <- 2 <- 3
8                9              8 <- 2 <- 3
*************************************************************/

/*************** OUTPUT-2 ********************
Enter the number of nodes: 4
Enter the source node    : 0

Enter the cost matrix:
0 1 4 999
1 0 2 6
4 2 0 3
999 6 3 0

Expected Output:
Node     Distance from source    Path
0         0                      0
1         1                      1 <- 0
2         3                      2 <- 1 <- 0
3         6                      3 <- 2 <- 1 <- 0
**********************************************************/
```

4. Using FIFOs as IPC channels, write a client – server program, the client sends the file name and the server sends back the requested text file if present.

```c
/* Server Program */
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>     // used for file handling
#include <sys/stat.h>  // used for mkfifo function
#include <sys/types.h> // mkfifo() has dependency on both types.h and stat.h

int main()
{
    char fname[50], buffer[1025];
    int req, res, n, file;

    mkfifo("req.fifo", 0777);
    mkfifo("res.fifo", 0777);

    printf("Waiting for request...\n");
    req = open("req.fifo", O_RDONLY);
    res = open("res.fifo", O_WRONLY);

    read(req, fname, sizeof(fname));
    printf("Received request for %s\n", fname);

    file = open(fname, O_RDONLY);
    if (file < 0)
    {
        write(res, "File not found\n", 15);
    }
    else
    {
        while ((n = read(file, buffer, sizeof(buffer))) > 0)
        {
            write(res, buffer, n);
        }
    }

    close(req);
    close(res);

    unlink("req.fifo");
    unlink("res.fifo");

    return 0;
}
```

```c
/* Client Program */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main()
{
    char fname[50], buffer[1025];
    int req, res, n;
    req = open("req.fifo", O_WRONLY);
    res = open("res.fifo", O_RDONLY);
    if (req < 0 || res < 0)
    {
        printf("Please Start the server first\n");
        exit(-1);
    }
    printf("Enter filename to request : ");
    scanf("%s", fname);
    // write file name to request file
    write(req, fname, sizeof(fname));
    printf("Received response\n");
    printf("-------------------------------------------\n");
    while ((n = read(res, buffer, sizeof(buffer))) > 0)
    {
        printf("%s", buffer);
    }
    printf("-------------------------------------------\n");

    close(req);
    close(res);
    return 0;
}
```

5. Using TCP/IP sockets, write a client – server program, the client sends the file name and the server sends back the requested text file if present.

```
/* Server Program */
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
    int server_fd, sock, fd, n;
    char buffer[1024], fname[50];
    struct sockaddr_in addr;

    /* creating socket file descriptor */
    /* sockfd = socket(domain, type, protocol) */
    server_fd = socket(AF_INET, SOCK_STREAM, 0);

    /* htons() converts the unsigned short integer
    ** from host byte order to network byte order.
    */
    addr.sin_family = AF_INET;
    addr.sin_port = htons(1234);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

```c
    /* attaching socket to port */
    /* bind(sockfd, addr , addrlen) */
    bind(server_fd, (struct sockaddr *)&addr, sizeof(addr));
    printf("\nServer is Online");

    /* listen for connections from the socket */
    /* listen(int sockfd, int backlog) */
    listen(server_fd, 5);

    /* accept a connection, we get a file descriptor */
    /* new_socket = accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen) */
    sock = accept(server_fd, NULL, NULL);

    /*  receive the filename */
    /* recv(int socket, const void *buffer, size_t length, int flags); */
    recv(sock, fname, 50, 0);
    printf("\nRequesting for file: %s\n", fname);

    /*  open the file in read-only mode */
    fd = open(fname, O_RDONLY);

    if (fd < 0)
    {
        send(sock, "\nFile not found\n", 15, 0);
    }
    else
    {
        while ((n = read(fd, buffer, sizeof(buffer))) > 0)
        {
            send(sock, buffer, n, 0);
        }
    }
    printf("\nFile content sent\n");
    close(fd);

    return 0;
}
```

```c
/* Client Program */
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <unistd.h>

int main()
{
    int sock, n;
    char buffer[1024], fname[50];
    struct sockaddr_in addr;

    /* socket creates an endpoint for communication */
    /* sockfd = socket(domain, type, protocol) */
    sock = socket(AF_INET, SOCK_STREAM, 0);
    addr.sin_family = AF_INET;
    addr.sin_port = htons(1234);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    /*  keep trying to esatablish connection with server */
    while (connect(sock, (struct sockaddr *)&addr, sizeof(addr)));

    printf("\nClient is connected to Server");

    printf("\nEnter file name: ");
    scanf("%s", fname);

    /* send the filename to the server */
    /* send(int socket, const void *buffer, size_t length, int flags); */
    send(sock, fname, sizeof(fname), 0);

    printf("\nRecieved file data\n");
    printf("---------------------------------------------------------\n");

    /*  keep printing any data received from the server */
    while ((n = recv(sock, buffer, sizeof(buffer), 0)) > 0)
    {
        buffer[n] = '\0';
        printf("%s", buffer);
    }

    printf("---------------------------------------------------------\n");

    return 0;
}
```

6. Using UDP, write a client – server program, to exchange messages between client and the server.

```
/* Server Program */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT 8080
#define MAX 1024

int main()
{
    int len, n;
    int sockfd;
    char buffer[MAX];
    char msg[MAX];
    struct sockaddr_in servaddr, cliaddr;
    // Creating socket file descriptor
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    // memset(void *address, int value, size_t length);
    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));
    len = sizeof(cliaddr);

    // Filling server information
    servaddr.sin_family = AF_INET;         // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY; // INADDR_ANY listen on all available
interfaces
    servaddr.sin_port = htons(PORT);

    // Bind the socket with the server address
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) >= 0)
    {
        printf("Waiting for message from client...\n");
    }

    while (1)
    {
        n = recvfrom(sockfd, (char *)buffer, sizeof(buffer), 0, (struct sockaddr
*)&cliaddr, &len);
        buffer[n] = '\0';
        printf("Client : %s", buffer);

        printf("Server : ");
        fgets(msg, MAX, stdin);
        sendto(sockfd, (const char *)msg, strlen(msg), 0, (const struct sockaddr
*)&cliaddr, len);
    }
    return 0;
}
```

```
/* Client Program */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT 8080
#define MAX 1024

int main()
{
    int n, len, sockfd;
    char buffer[MAX];
    char msg[MAX];
    struct sockaddr_in servaddr;

    // Creating socket file descriptor
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    connect(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr));

    while (1)
    {
        printf("Client : ");
        fgets(msg, MAX, stdin);
        sendto(sockfd, (const char *)msg, strlen(msg), 0, (const struct sockaddr
*)&servaddr, sizeof(servaddr));

        n = recvfrom(sockfd, (char *)buffer, sizeof(buffer), 0, NULL, NULL);
        buffer[n] = '\0';
        printf("Server : %s", buffer);
    }
    return 0;
}
```