DAYANANDA SAGAR COLLEGE OF ENGINEERING
BANGALORE-78
DEPARTMENT OF TELECOMMUNICATION ENGINEERING

VII SEMESTER

COMPUTER COMMUNICATION & NETWORKING LABORATARY (17TE7DECCN) MANUAL

Prepared by.

NAGARATHNA,
Assistant Professor,
Department of Telecommunication Engineering.

DO'S AND DON'TS IN THE LABORATORY

- **Come in proper dress code as prescribed by the college**
- **Read the theory of the experiment to be performed well in advance and understand the working principle clearly before conducting the experiment.**
- **Datasheets have to be written in the lab.**
- **Come prepared for viva - voce**
- **Follow the given procedure in the same sequence.**
- **Handle the equipment with care.**
- **After the completion of the experiment get the datasheets signed by the staff-in-charge.**
- **Make sure to switch of all the systems and return equipments to the stores after the completion of the experiment.**

Note:
Lab manual should be treated as a guideline only.

COMPUTER COMMUNICATION & NETWORKING LABORATARY

SYLLABUS

I) CCN Programming using C/C++(3 lab sessions of 3hrs each)

PART-A

1. Simulate bit stuffing & de-stuffing using HDLC
2. Simulate character stuffing & de-stuffing using HDLC
3. Simulate the Shortest Path Algorithm
4. Encryption and Decryption of a given message using Substitution method
5. Encryption and Decryption of a given message using Transposition method
6. Find Minimum Spanning Tree of a graph
7. Compute Polynomial Code Checksum for CRC-CCITT.

PART-B

1. Asynchronous and Synchronous Communication using RS232 / Optical Fiber / Twisted Pair / RJ45

2. Using fork function, create TWO processes and communicate between them

3. Communicate between TWO PCs, using simple socket function.

PART-A

1. Simulate bit stuffing & de-stuffing using HDLC

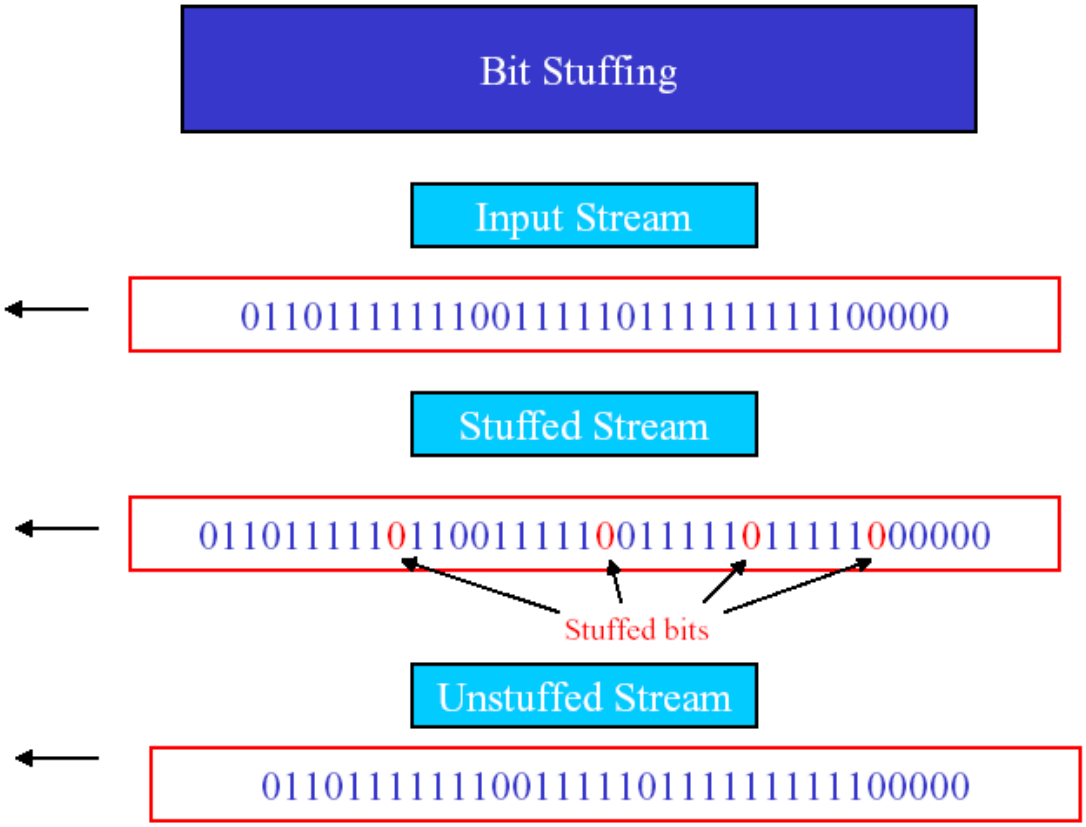
Framing involves identifying the beginning and end of a block of information within a digital stream. In asynchronous data transmission, since transmissions do not occur at regular intervals, the receiver resynchronizes at the start of each eight bit character by the use of a start bit that precedes and a stop bit that ends each character .In synchronous data transmission bits are transmitted at regular intervals and the receiver has the circuitry that recovers and tracks the frequency and bit transitions of the received data Framing may involve delineating the boundaries between frames that are of fixed length or it may involve delineating between frames that are of variable length. Variable length frames need more information to delineate. The methods available include:

Special characters to identify beginning and end of frame

Special bit patterns-"flags" to identify the beginning and end of frames and character counts

Bit Stuffing:

Flag based synchronization was developed to transfer an arbitrary number of bits within a frame. The figure below shows the structure of an HDLC frame. The beginning and end of an HDLC frame is indicated by the presence of an eight bit flag. The flag in HDLC consists of the byte 01111110 that is HEX 7E . Bit stuffing prevents the occurrence of the flag inside the frame. The transmitter examines the contents of the frame and inserts an extra 0 after each instance of five consecutive 1s. The transmitter then attaches the flag at the beginning and end of the resulting bit stuffed frame. The receiver looks for five consecutive 1s in the received sequence. Five 1s followed by a 0 indicate that the 0 is a stuffing bit and so the bit is removed. Five consecutive 1s followed by 10 indicate a flag. Five 1 s followed by 11 indicate an error. The example below show bit stuffing in HDLC



HDLC Frame

The high level data link control protocol is a bit oriented protocol which uses bit stuffing for data transparency. HDLC frame is as shown.

8	8	8	>0	16	8
0111 1110	Address	Control	Data	Checksum	0111 1110

The *Control field* is used for sequence numbers and acknowledgements and other purposes. The *Data field* may contain arbitrary information. The *Checksum field* is a minor variation of the CRC code using CRC-CCITT as the generator. Frames are delimited by 0111 1110 and this sequence is not allowed in the data field.

Algorithm for Bit Stuffing

1. Input data sequence
2. Add start of frame to output sequence
3. for every bit in input

a. Append bit to output sequence

b. Is a bit 1?

Yes:

Increment count

If count is 5, append 0 to output sequence and reset count

No:

Set count to 0

4. Add stop of frame bits to output sequence.

Algorithm for Bit Destuffing

1. Input data sequence
2. Remove start of frame to output sequence
3. for every bit in input

a. Append bit to output sequence

b. Is a bit 1?

Yes:

Increment count

If count is 5, remove 0 from input sequence and reset count

No:

Set count to 0

4. Remove stop of frame bits to output sequence.

/ Program to simulate bit stuffing & destuffing where the flag byte is 01111110 */*

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

void main()
{

    char ch, array[50]={"01111110"},recd_array[50];
    int counter=0,i=8,j,k;
    clrscr();

    printf("Enter the original data stream for bit stuffing : \n");
    while((ch=getche())!='\r')
    {
        if (ch=='1')
                ++counter;
```

```
else
                counter=0;
        array[i++]=ch;
        if (counter==5)    /* If 5 ones are encountered append a zero */
        {
                array[i++]='0';
                counter=0;
        }
    }
array[i]='\0';
strcat(array,"01111110");
    array[i+8]='\0';

    printf("\nThe stuffed data stream is : %s \n",array);

    /* Destuffing */
    counter=0;
    printf("\nThe destuffed data stream is : \n");
    for (j=8,k=0;j<(strlen(array)-8);++j)
    {
        if (array[j]=='1')
                ++counter;
        else
                counter=0;
        recd_array[k++]=array[j];

         if (counter==5)  /* If five ones appear, delete the following zero */
        {
                ++j;
                counter=0;
        }
    }
recd_array[k]='\0';

printf(" the destuffed data stream is %s",recd_array);
    getch();
}
```

Output:

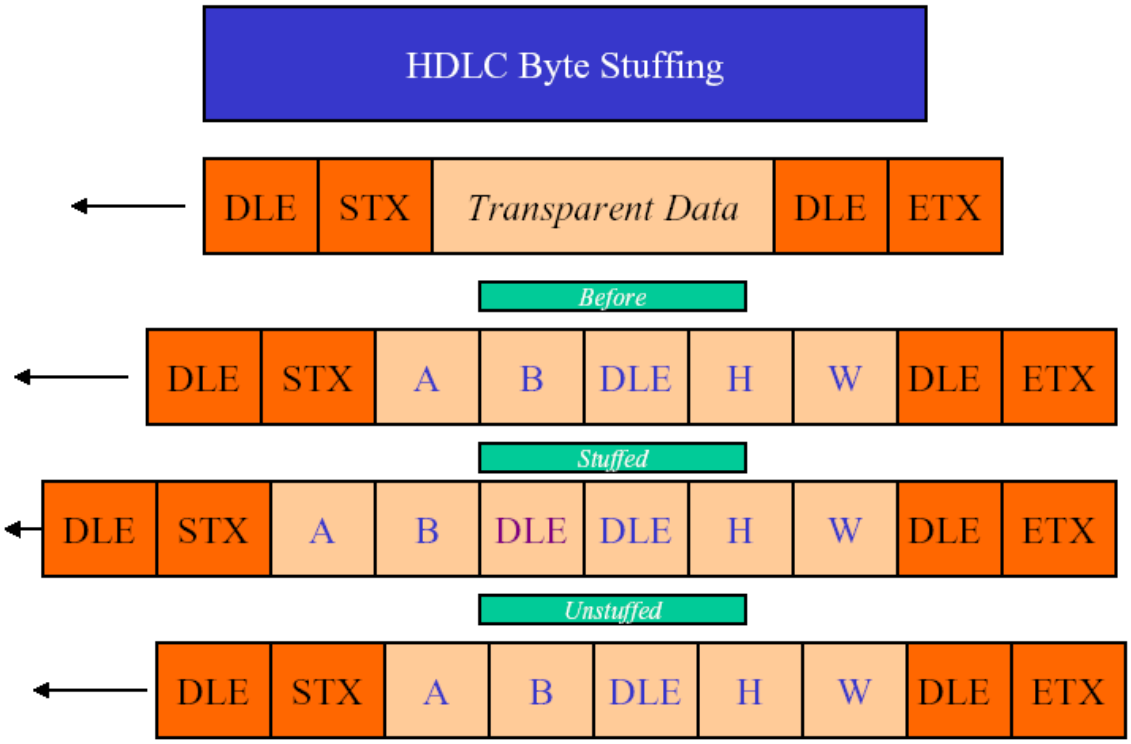
```
Enter the original data stream for bit stuffing :
011011111111110010
The stuffed data stream is :
011111100110111110111110001001111110
The destuffed data stream is :
011011111111110010
```

2. Simulate character (byte) stuffing & de-stuffing using HDLC

Character based frame synchronization methods are used when the information in a frame consists of an integer number of characters. For example, asynchronous transmission systems are used extensively to transmit sequences of printable characters using 8 bit ASCII code . To delineate a frame of characters, special eight bit codes that do not correspond to printable characters are used as control characters. In ASCII code all characters with hexadecimal values less than 20 correspond to non printable characters.(refer the ASCII table shown in figure.)

In particular STX (start of text) control character has hex value 02 and indicates the beginning of a frame and an ETX (end of text) character has HEX value 03 and denotes the end of a frame. This method works if the frame contains only printable characters. If a frame carries computer data, then it is possible that an ETX character will appear inside the frame and cause the receiver to prematurely truncate the frame. This method is not transparent because the frame cannot carry all possible bit sequences. The use of byte stuffing enables transparent operation. Byte stuffing operates as follows. A special DLE (data link escape) control with hex value 10 is introduced. The two character sequence DLESTX is used to indicate the beginning of a frame and DLE ETX denotes the end of a frame. The receiver looks for these character pairs to identify the beginning and end of frames. In order to deal with the occurrence of DLE STX or DLE ETX in the data contained in the frame, an extra DLE is inserted or stuffed before the occurrence of a DLE inside the frame.

Consequently every legitimate DLE in the data is replaced by two DLEs. The only incidence of an individual DLE occurs when DLE precedes the STX or the ETX that identify the beginning and end of the frame.



Decimal	Hexadecimal	ASCII	DESCRIPTION
0	00	NUL	CTRL/1
1	01	SOH	CTRL/A
2	02	STX	CTRL/B
3	03	ETX	CTRL/C
4	04	EOT	CTRL/D
5	05	ENQ	CTRL/E
6	06	ACK	CTRL/F
7	07	BEL	CTRL/G
8	08	BS	CTRL/H, BACKSPACE
9	09	HT	CTRL/I, TAB
10	0A	LF	CTRL/J, ENTER
11	0B	VT	CTRL/K
12	0C	FF	CTRL/L
13	0D	CR	CTRL/M, RETURN
14	0E	SO	CTRL/N
15	0F	SI	CTRL/O
16	10	DLE	CTRL/P

Algorithm for Character Stuffing/Destuffing

1. Input data sequence

2. Add start of frame to output sequence (DLE STX) /remove start of frame chars from input sequence.

3. for every character in input

   a. Append character to output sequence

   b. Is character DLE?

      Yes: Add DLE to output sequence/remove next DLE from input sequence.

4. Add stop of frame chars to output sequence/remove stop of frame chars from input sequence.

/ Program to demonstrate character stuffing */*

```
# include <stdio.h>
# include <conio.h>
# define DLE 16
# define STX 2
# define ETX 3

void main()
{
    char ch;
    char array[50]={DLE,STX};
     char destuff[50];
    int i=2,j,k;
    clrscr();

    printf("Enter the data stream (Ctrl+B->STX, Ctrl+C->ETX, Ctrl+P->DLE) : \n");
    while ((ch=getch())!='\r')
    {
        if (ch==DLE)
        {
                array[i++]=DLE;
                printf("DLE ");
        }
        else if (ch==STX)   printf("STX ");
        else if (ch==ETX)   printf("ETX ");
        else   printf("%c ",ch);
        array[i++]=ch;
    }
    array[i++]=DLE;
    array[i++]=ETX;
    printf("\nThe stuffed stream is \n");
    for (j=0;j<i;++j)
    {
        if (array[j]==DLE)  printf("DLE ");
        else if (array[j]==STX)   printf("STX ");
        else if (array[j]==ETX)   printf("ETX ");
        else     printf("%c ",array[j]);
    }
```

```
    k=0;
    printf("\nThe destuffed data stream is : \n");
    for (j=2;j<i-2;++j)
    {
       destuff[k++]=array[j];
        if (array[j]==DLE)
        {       printf("DLE ");
                ++j;
        }
        else if (array[j]==STX)
                printf("STX ");
        else if (array[j]==ETX)
                printf("ETX ");
        else
                printf("%c ",array[j]);
    }
    getch();
}
```

Output:

Enter the data stream (Ctrl+B->STX, Ctrl+C->ETX, Ctrl+P->DLE) :
A DLE B
The stuffed stream is
DLE STX A DLE DLE B DLE ETX
The destuffed data stream is :
A DLE B

Enter the data stream (Ctrl+B->STX, Ctrl+C->ETX, Ctrl+P->DLE) :
*ABCDLEXYZ
The stuffed stream is
DLE STX ABCDLEXYZ DLEETX
The destuffed data stream is :
ABCDEXYZ

*Note: If you type DLE then stuffing will not be done. DLE will be stuffed only when you press Ctrl+P.

2. Simulate the Shortest Path Algorithm

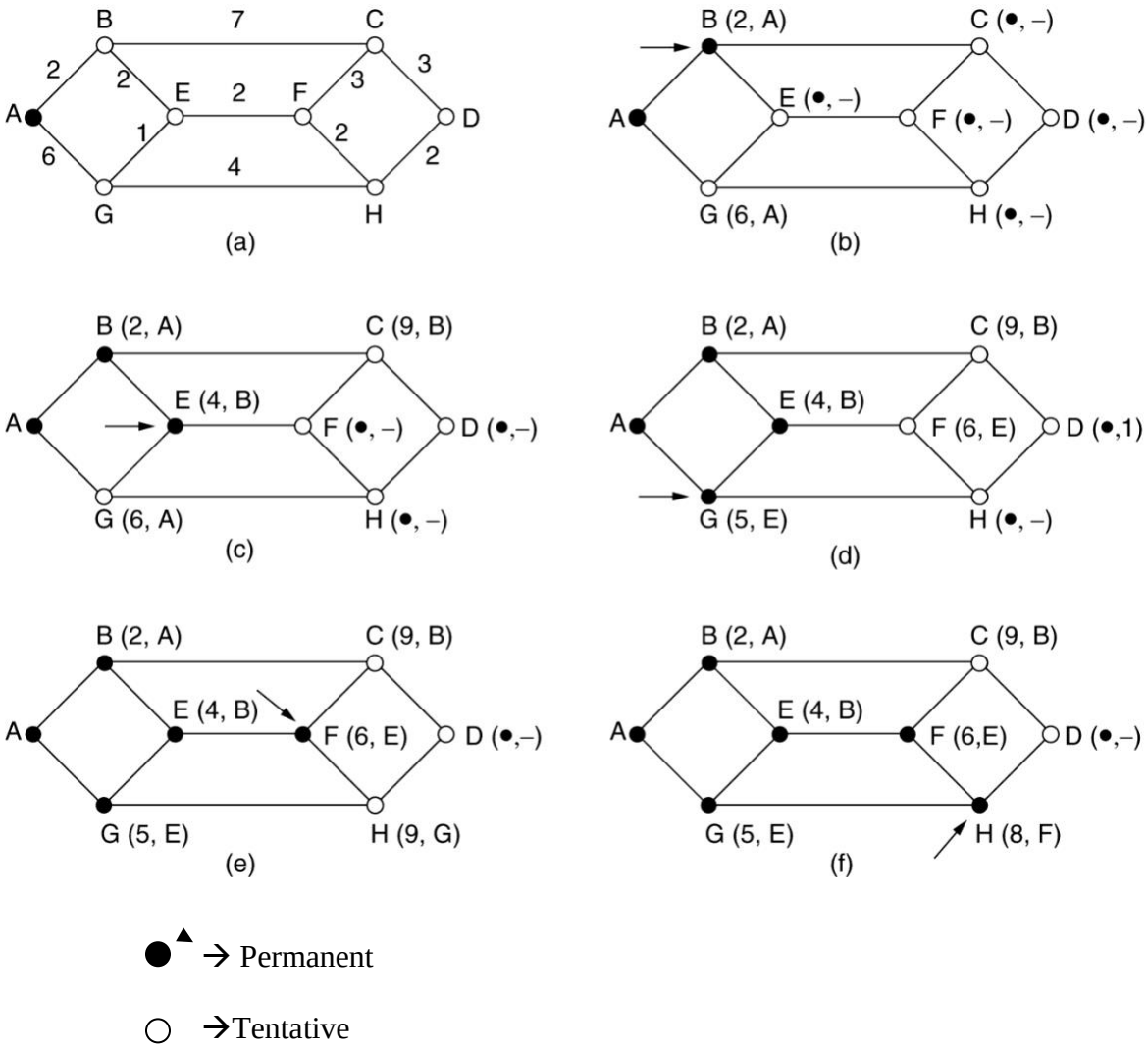
. In order to transfer packets from a source host to the destination host , the network layer must determine the path or route that the packets are to follow. This is the job of the network layer routing protocol. At the heart of any routing protocol is the routing algorithm that determines the path for a packet from source router to destination router. Given a set of routers, with links connecting the routers, a routing algorithm finds a good path from source router to destination router, according to some cost criterion. These can be,

1. Hop count: 1/ capacity: The cost is inversely proportional to the link capacity . One assigns higher costs to lower capacity links . The objective is to send a packet through a path through a path with the highest capacity. If each link has equal capacity , then the shortest path is the path with the minimum number of hops
2. Transmission speed: The speed at which the various links operate is an important part of a route's efficiency. Faster links obviously take precedence over slow ones.
3. Congestion: Network congestion caused by the current traffic pattern is considered when evaluating a route and links that are overly congested are bypassed
4. Route cost : The route cost is a metric assigned by the network administrator used to rate the relative usability of various routes . The cost can refer to the literal financial expense incurred by the link or any other pertinent factor.
5. Packet delay: The cost is proportional to an average packet delay which includes queuing delay in the switch buffer ad propagation delay in the link. The shortest path represents the fastest path to reach the destination.

Dijkstra's method of computing the shortest path or shortest path routing is a static routing algorithm. It involves building a graph of the subnet, with each node of the graph representing a router and each arc representing a communication line or a link. To find a route between a pair of routers, the algorithm just finds the shortest path between them on the graph.

In Dijkstra's algorithm the metric used for calculation is distance. Each node is labeled with its distance from the source node along with the previous node in the path. Initially no paths are known so all nodes are labeled with infinity. As the algorithm proceeds and paths are found, the labels may change, reflecting better paths. A label may either be tentative or permanent. Initially all nodes are tentative and once it is discovered that the shortest possible path to a node is got it is made permanent and never changed after that.

Exmaple:



● → Permanent

○ →Tentative

(fig: The first 5 steps used in computing shortest path from A to D. The arrows indicates working node.)

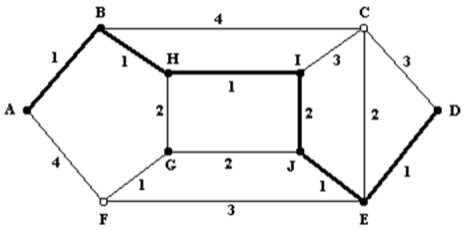
To illustrate how the algorithm works, refer the weighted graph shown in figure. We want to find the shortest path from A to D. We start out by marking node A as permanent, indicated by a filled-in circle (as A is the source) . Then we examine, in turn each of the nodes which are tentative (shown by unfilled circles) adjacent to A(the working node), relabeling each one with the distance to A. Whenever a node is relabeled, we also label it with the node from which probe was made so that we can reconstruct the final path later.

Having examined each of the nodes adjacent to A, we examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent, as shown in fig b. This one becomes the new working node. We now start at B and examine all nodes

adjacent to it. If the sum of the label on B and the distance from B to the node being considered is less than the label on that node we have a shorter path, soothe node is relabeled. After all the nodes adjacent to the working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively-labeled node, with smallest weight. This node is made permanent and becomes the working node for the next round.

Algorithm

1. Input graph data
2. Make all nodes TENTATIVE.
2. Input source and destination
3. Make source, the working node
4. Make the working node PERMANENT.
5. Check all tentative nodes, which are connected to working node. Update weight if required.
6. Find TENTATIVE node with smallest weight. Make this the new working node.
7. If working node is destination, go to step 8 else go to step 4
8. Trace back from destination to source.




```
# include<stdio.h>
# include<conio.h>
/*Total number of nodes in network*/
# define N 10
/*State of each node*/
# define PERMANENT 1
# define TENTATIVE  0


struct node
{
        int weight;
         int prev;
         int state;
};

void main()
```

```
{
        int table[N][N] =
                        /* A B C D E F G H I J*/
                /*A*/     {  {0,1,0,0,0,4,0,0,0,0},
                /*B*/        {1,0,4,0,0,0,0,1,0,0},
                /*C*/        {0,4,0,3,2,0,0,0,3,0},
                /*D*/        {0,0,3,0,1,0,0,0,0,0},
                /*E*/        {0,0,2,1,0,3,0,0,0,1},
                /*F*/        {4,0,0,0,3,0,1,0,0,0},
                /*G*/        {0,0,0,0,0,1,0,2,0,2},
                /*H*/        {0,1,0,0,0,0,2,0,1,0},
                /*I*/        {0,0,3,0,0,0,0,1,0,2},
                /*J*/        {0,0,0,0,1,0,2,0,2,0}
                        };/* A B C D E F G H I J*/
                            /*interpret as A is connected to B at a weight of 1 as
                            table[A][B]=table[B][A]=1*/
        int src,dest,i,w_node;
                int min;

        struct node nodes[N];

        for(i=0;i<N;++i)
        {
                nodes[i].state=TENTATIVE;
                nodes[i].weight=10000;
        }

        printf("\nEnter Source:");
        src=getche();

        w_node=src=toupper(src)-'A';
                nodes[src].prev=-1;
        nodes[src].weight=0;
        printf("\nEnter Destination:");
        dest=toupper(getche())-'A';
        do
        {
                nodes[w_node].state=PERMANENT;

        for(i=0;i<N;++i)
          {
                        if(table[w_node][i]!=0 && nodes[i].state==TENTATIVE)
                {

                if(nodes[w_node].weight+table[w_node][i]<nodes[i].weight)
                        {
nodes[i].weight=nodes[w_node].weight+table[w_node][i];
                nodes[i].prev=w_node;

                        }
} }
 /*Find minimum weighted Tentative node*/
min=10000;
                for(i=0;i<N;++i)
                {
                        if(nodes[i].state==TENTATIVE && nodes[i].weight<min)
                        {
                                min=nodes[i].weight;
                                w_node=i;
                        }
```

```
            }
        } while(w_node!=dest);

        printf("\nShortest Path got--->\n%c",dest+65);
        do
        {
                w_node=nodes[w_node].prev;
                printf("<-%c",w_node+65);
        } while(w_node!=src);
        printf("\nAt a total weight of:%d",nodes[dest].weight);
}
```

Output:

1. Enter Source:A
   Enter Destination:D
   Shortest Path got--->
   D<-E<-J<-I<-H<-B<-A
   At a total weight of: 7

2. Enter the Source: G
   Enter the Destination:C
   Shortest Path got-→
   G<-J<-E<-C
   At a total weight of: 5

3. Encryption and Decryption of a given message

The science and art of manipulating messages to make them secure is called Cryptography. An original message to be transformed is called the plain text and the resulting message after the transformation is called the cipher text. The process of converting the plain text into cipher text is called encryption. The reverse process is called decryption. The algorithm used for encryption and decryption is often called a cipher Substitution ciphers are a common technique for altering messages in games and puzzles. Each letter of the alphabet is mapped into another letter. The cipher text is obtained by applying the substitution defined by the mapping to the plain text. Transposition ciphers are another type of encryption scheme. Here the order in which the letters appear is altered. The letters may be written into an array in one order and read out in a different order.

In cryptography, the messages to be encrypted; known as plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process, known as cipher text, is then transmitted, often by messenger or radio. We assume that the enemy or intruder, hears and accurately copies down the complete cipher text. However, unlike the intended recipient, he does not know what the decryption key is and so cannot decrypt the cipher text easily. Sometimes the intruder cannot only listen to the communication channel (passive intruder) but can also record messages and play them later, inject his own messages, or modify legitimate messages before they get to the receiver (active intruder). The art of breaking ciphers is called cryptanalysis. The art of devising ciphers (cryptography) and breaking them (cryptanalysis) is collectively known as cryptology. It will often be useful to have a notation for relating plaintext, ciphertext and keys. We will use C=EK(P) to mean that the encryption of the plaintext P using key K gives the ciphertext C. Similarly, P=DK(C) represents decryption of C to get the plaintext again. It then follows that

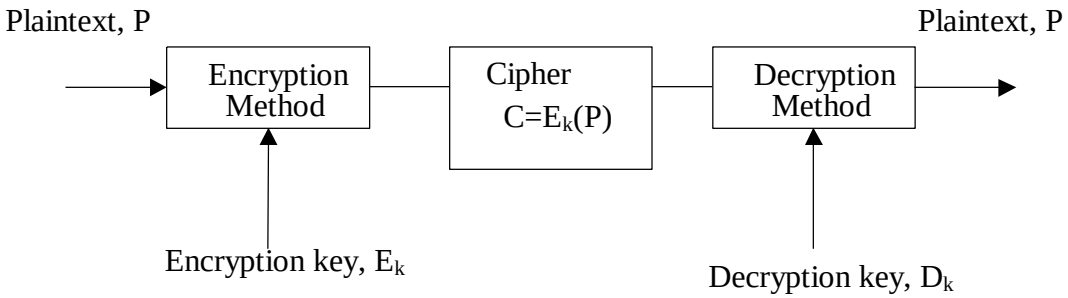
$$DK(EK(P)) = P$$

Fundamental rule of cryptography is that one must assume that the cryptoanalyst knows the general method of encryption used. Encryption methods are historically divided into two categories: substitution ciphers and transposition ciphers.

Encryption Method Cipher C=Ek(P)

Decryption Method Plaintext P=DK(EK(P))

Encryption key-Ek, Decryption key-Dk, Plaintext-P



$D_k(E_K(P)) = P$

Substitution Ciphers

In a substitution cipher, each letter or group of letters is replaced by another letter by adding key to disguise it.

Plaintext: a b c d e f g h i j k l m n o p q r s t u v w x y z

Key=2

Ciphertext: c d e f g h i j k l m n o .....

Example : attack
Key=3
Encrypted data: dwwdfo

Algorithm for Encryption by Substitution Method

1. Read data to be encoded
2. for every character of data
3. add the key to the character.
4. Print encrypted data.
5. subtract the key from character
6. print the decrypted data.

/* Encryption by Substitution method */

/* Program to encrypt given data even numbers sequence is a key on which you change data change all small letters to big and vice versa*/

```
# include<string.h>
# include<ctype.h>
# include<stdio.h>
void main()
{
char data[50];
char encoded[50];
char decoded[50];
int i,len,key;
printf("\nEnter data to be encoded:");
gets(data);
len=strlen(data);
printf("\nenter the key for encryption\n");
scanf("%d",&key);
for(i=0;i<len;i++)
encoded[i]= 'a'+((data[i]-'a')+key)%25;
 encode[i]='\0';
printf("\n the encrypted data is %s", encoded);
 /*decryption */



for(i=0;i<len;i++)
{
decoded[i]='a'+( (( encoded[i]-'a')+(26-key))%26)
}

decoded[i]='\0';
printf("the decrypted message is %s", decoded);

}
```

Output:

Enter data to be encoded: hello world
Enter the key for encryption:2
Encrypted data: jgnnq
Decrypted message : hello world

Transposition Ciphers

Substitution ciphers preserve the order of the plaintext symbols but disguise them. Transposition ciphers, in contrast, reorder the letters but do not disguise them. The figure below depicts a common transposition cipher, the columnar transposition. The cipher is keyed by a word or phrase not containing any repeated letters. In this example, MEGABUCK is the key. The purpose of the key is to number the columns, column 1 being under the letter closet to the start of the alphabet and so on. The plain text is written horizontally, in rows. The cipher text is read out by columns, starting with the column whose key letter is the lowest.

Example:

Plain text : please transfer one million dollars to my swiss bank account six six two

Key text:	**M**	**E**	**G**	**A**	**B**	**U**	**C**	**K**
	7	**4**	**5**	**1**	**2**	**8**	**3**	**6**
	p	l	e	a	s	e	-	t
	r	a	n	s	f	e	r	-
	o	n	e	-	m	i	l	l
	i	o	n	-	d	o	l	l
	a	r	s	-	t	o	m	y
	-	s	w	i	s	s	-	b
	a	n	k	-	a	c	c	o
	u	n	t	-	s	i	x	-
	s	i	x	-	t	w	o	-

Ciphertext:

as---o---sf,dtsast-rllm-cxolanorsnnienenswktxt-llybo—proia-auseeioosciw

Algorithm for Encryption and Decryption by Transposition Method

1. Get the plain text
2. replace spaces with '–'
3. add '–' at the end until the length of the plain text is a multiple of length of the key.
4. Set the character='A'
5. search for character in key text, index=0
6. if found

    a.)yes: pick character from plain text at index position store it in cipher text

    increment index by length of key text repeat 6a

b) no: increment index, if index=length of key text go to step7 else repeat 6.

7. increment character, check character='Z', yes: go to step 8, no: repeat 6.

8. Get the encoded text

9 replace '-'with spaces

10. set the character='A'

11. Search for character in key text, index=0

12. if found

a.)yes: pick character from cipher(encoded) text at index position store it in plain text ,increment index by length of key text repeat 12a

b) no: increment index, if index=length of key text go to step13 else repeat 12.

13. Increment character, check character='Z', yes: go to step 14, no: repeat 12.

14. Terminate

```
/*program to encrypt and decrypt message using transposition method */
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h>


void main()
{

char inp[50],out[50],key[20], dec[50];
int i,j,k=0,ilen,klen,t;
char ch='A';
clrscr();
printf("Enter the string to be encrypted:");
gets(inp);
printf("\n enter the key string:");
gets(key);
ilen=strlen(inp);
klen=strlen(key);

for(t=0;t<ilen;t++)
     if(inp[t]==' ')
       inp[t]='-';

while(ilen%klen)
inp[ilen++]='-';

for(ch='A';ch<='Z';ch++)
  for(j=0;j<=klen;j++)
    if(toupper(key[j])==ch)
       for(i=j;i<ilen;i+=klen)
          out[k++]=inp[i];

out[k]='\0';
printf("\n the encrypted output is : %s', out);
```

```
/* decryption */

printf("enter the key text\n");
gets(key);

for(ch='A',k=0; ch<='Z';ch++)
 for(j=0;j<klen;j++)
   if(toupper(key[j])==ch)
    for(i=j;i<ilen;i+=klen)
      dec[i]=out[k++];

dec[k]='\0';

for(t=0;t<ilen;t++)
  if(dec[t]=='-')
     dec[t]=' ';


printf("\n \n the decrypted output is : %s", dec);
getch();

}
```

Output:

1. Enter the string to be encrypted: please transfer one million dollars to my swiss bank account six six two
Enter the key string: megabuck
The encrypted message: as---i---sfmdtsast-rllm-cxolanorsnnienenswktxt-llybo—proia-auseeiioosciw
(refer example)

2. Enter the string to be encrypted: CCN Lab is very easy
Enter the key string : TELECOM
The encrypted message: le-csa-vym-sbu-ar-cie

5. Find Minimum Spanning Tree of a graph

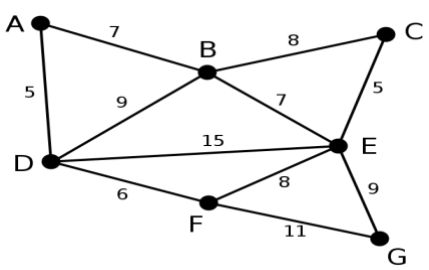
Given a connected, undirected graph, a spanning tree of that graph is a subgraph which is a tree and connects all the vertices together. A single graph can have many different spanning trees. For instance, a complete graph of 4 nodes can have 16 spanning trees. We can also assign a *weight* to each edge, which is a number representing how unfavorable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. A **minimum spanning tree** (**MST**) or **minimum weight spanning tree** is then a spanning tree with weight less than or equal to the weight of every other spanning tree.

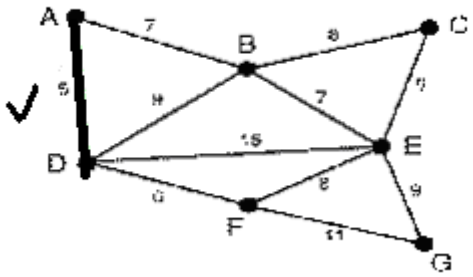
There may be several minimum spanning trees of the same weight having a minimum number of edges in particular, if all the edge weights of a given graph are the same, then every spanning tree of that graph is minimum. If there are *n* vertices in the graph, then each tree has *n*-1 edges.

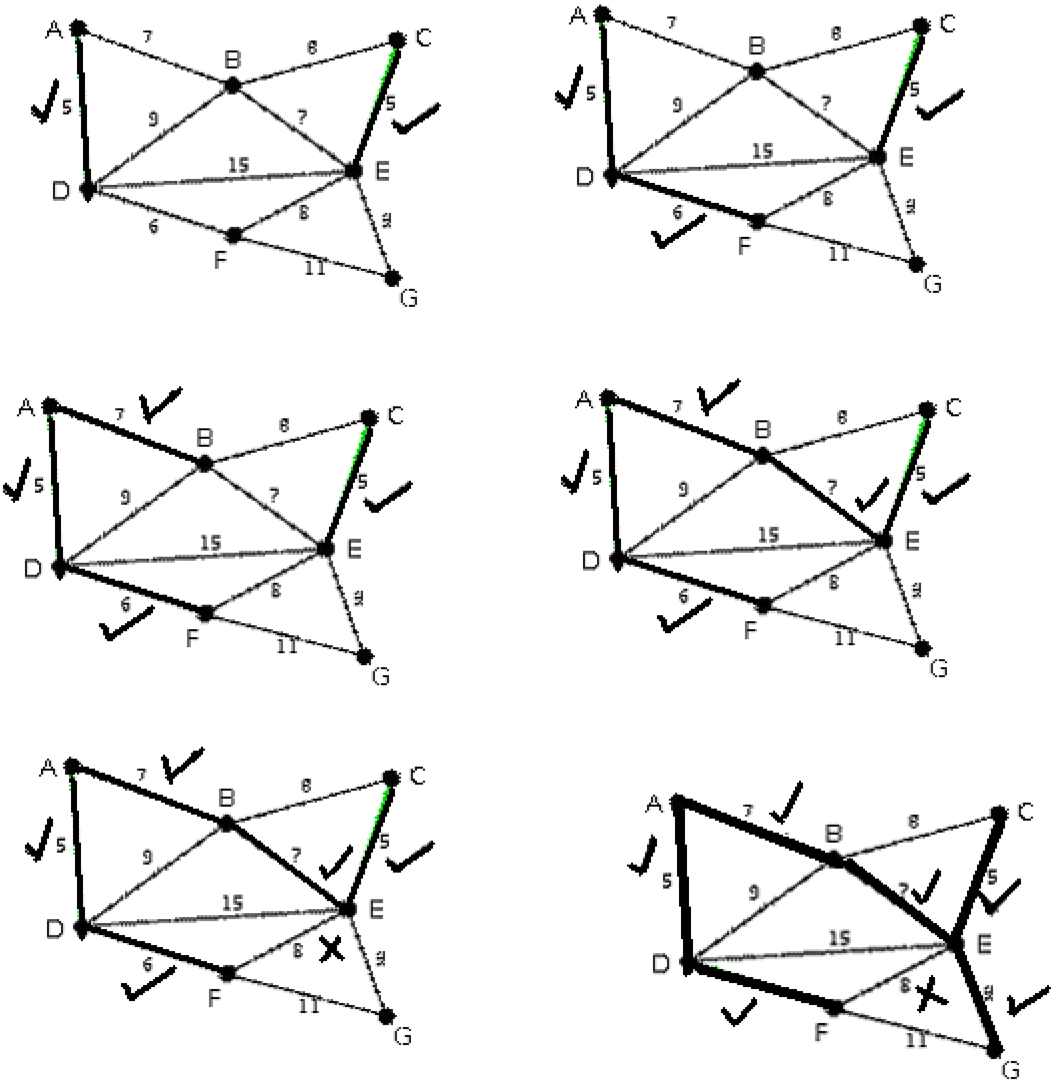
.A spanning tree includes all the routers but contains no loops. If each router knows which of its lines belong to the spanning tree, it can copy an incoming broadcast packet onto all the spanning tree lines except the one it arrived on. This method makes excellent use of bandwidth, generating the absolute minimum number of packets necessary to do the job. The only problem is that each router must have knowledge of some spanning tree for it to be applicable. Sometimes this information is available (e.g. with link state routing) but sometimes it is not (e.g. with distance vector routing).

There are now two algorithms commonly used, Prim's algorithm and Kruskal's algorithm. The following example explains Kruskal's algorithm to construct minimum spanning tree.

Example:







✓ → Selected ✗ → Not Selected.

In order to construct a minimum spanning tree for the network shown in fig1, first select all the edges according to their weights(least should be selected first.), chek whether the selection does not create a loop. (refer fig2 to 7). Continute this process till number edges selected is n-1. Where ‘n’ is the number of nodes in a network. The minimum spanning tree is shown in fig8.

Note: Number of edges in the minimum spanning tree will be n-1.

Finding the minimum spanning tree

The easiest to understand and probably the best one for solving problems by hand is Kruskal’s algorithm.

1. Input number of vertices
2. Input the edge weights

3. Sort the edge weights in ascending order

4. Pick the lowest edge and join the corresponding vertices

5. Repeat till edges are marked making sure that there are no closed loops

   (number of vertices –1)

6. Display selected edges.

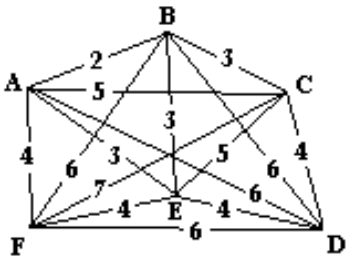
NOTE: In the program given, the variable 'set' is used to eliminate closed loops. Before joining two nodes, each node's 'set' is checked and if found to be the same, the join is not performed. If the nodes are of different 'sets' then they are joined and then made of the same set.

```
/* Program for finding the Minimum Spanning Tree of a network */
#include<stdio.h>
#include<conio.h>
struct edge
{
int fn,sn,dist;
}e[50],temp;

void main()
{
int i,j,n,set[10],ecnt=0,m,k;
clrscr();
printf("enter the no of nodes\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
  for(j=i+1;j<n;j++)
   {
printf("enter the distance between %c and %c \n",i+'A',j+'A');
scanf("%d",&e[ecnt].dist);
e[ecnt].fn=i;
e[ecnt].sn=j;
ecnt++;
  }
}

for(i=0;i<n;i++)
{
set[i]=i;
}

for(i=0;i<ecnt;i++)
{
for(j=0;j<ecnt-i-1;j++)
{
if(e[j].dist>e[j+1].dist)
{
temp=e[j];
e[j]=e[j+1];
e[j+1]=temp;
}
}
```

```
}
printf("\n the spanning tree is\n");

for(i=0;i<ecnt;i++)
{
m=set[e[i].fn];
k=set[e[i].sn];
     if(m!=k)
     {
         printf(" %c-->%c, distance=%d",e[i].fn+'A',e[i].sn+'A',e[i].dist);
          for(j=0;j<n;j++)
          {
          if(set[j]==k)
          {
          set[j]=m;
          }
          }


     }

}



getch();
}
```

Output:

Enter the number of vertices : 6
Enter distance between vertex A and B : 2
Enter distance between vertex A and C : 5
Enter distance between vertex A and D : 6
Enter distance between vertex A and E : 3
Enter distance between vertex A and F : 4
Enter distance between vertex B and C : 3
Enter distance between vertex B and D : 6
Enter distance between vertex B and E : 3
Enter distance between vertex B and F : 6
Enter distance between vertex C and D : 4
Enter distance between vertex C and E : 5
Enter distance between vertex C and F : 7
Enter distance between vertex D and E : 4
Enter distance between vertex D and F : 6
Enter distance between vertex E and F : 4



Minimal Spanning Tree :

A <----> B Distance : 2
A <----> E Distance : 3
B <----> C Distance : 3
A <----> F Distance : 4
C <----> D Distance : 4

5. Compute Polynomial Code Checksum for CRC-CCITT

The polynomial code (also known as a cyclic redundancy code or CRC code) is widely used. Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^{k-1} to x^0. Such a polynomial is said to be of a degree k-1. The high-order (left most) bit is the coefficient of x^{k-1}; the next bit is the coefficient of x^{k-2}, and so on. For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1,1,0,0,0 and 1 : $x^5 + x^4 + x^0$.

When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, G(x), in advance. Both the high and low-order bits of the generator must be 1. To compute the checksum for some frame with m bits, corresponding to the polynomial M(x), the frame must be longer than the generator polynomial. The idea is to append a checksum to the end of the frame in such a way that the polynomial represented by the check summed frame is divisible by G (x). When the receiver gets the checksummed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

The algorithm for computing checksum is as follows:

1. Let r be the degree of G(x). Append r zero bits to the low-order end of the frame, so it now contains m+r bits and corresponds to the polynomial $x^r M(x)$.
2. Divide the bit string corresponding to G(X) into the bit string corresponding to $x^r M(x)$ using modulo-2 division.
3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^r M(x)$ using modulo-2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial T(x).

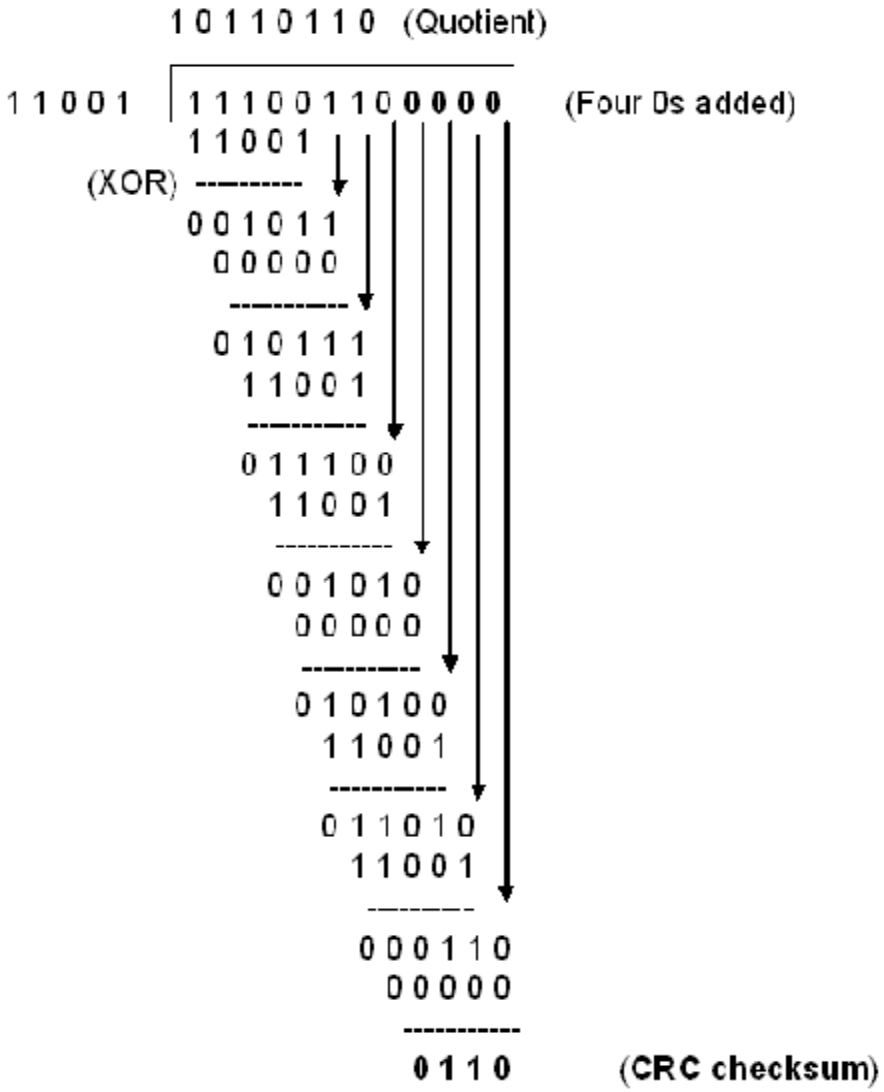
The CRC-CCITT polynomial is $x^{16} + x^{12} + x^5 + 1$

Example to calculate CRC:

Data Frame to be transmitted: 11100110

Generator Polynomial: 11001 ($x^4 + x^3 + 1$)

Message after appending four zeroes (as degree of polynomial is 4): 111001100000



Transmitted Frame= Message frame+ CRC checksum(Remainder)

= 111001100110.

Error Detection:

The receiver divides the message (including the calculated CRC), by the same polynomial the transmitter used. If the result of this division is zero, then the transmission was successful. However, if the result is not equal to zero, an error occurred during the transmission.

Assume for the previous example if the received frame is: **111001100110**
The received frame is divided by the same generator polynomial as used by the transmitter i.e., **11001 (**$x^4 + x^3 + 1$), we get remainder **0000,** indicating, **"No Transmission Error".**
Else if the remainder is not 0000 indicates **"Transmission error"**

/ C Program to compute polynomial code checksum */*

```
#include<stdio.h>
#include<math.h>
#include<conio.h>
#include<string.h>
#define degree16
int result[50];
int gen[17]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};   /ccitt polynomial * x16 + x12 + x5 + 1*/

int length;
void calcrc();
void main()
{
int array[50],flag=0;
int i=0;
char ch;
clrscr();
printf("enter frame ");
while((ch=getche())!='\r')
{
array[i++]=ch-'0';

}
length=i;
for(i=length;i<length+degree;i++)
{
array[i]=0;
}
length=length+degree;
for(i=0;i<length;i++)
{
result[i]=array[i];

}
calcrc();

for(i=0;i<length;++i)
{
result[i]=(result[i]|array[i]);
```

```
}

printf("\n the result is \n");
for(i=0;i<length;i++)
printf(" %d",result[i]);

getch();

printf("\nEnter the stream for which CRC has to be checked : ");
    i=0;


    while((ch=getche())!='\r')
        array[i++]=ch-'0';

    length=i;

    for (i=0;i<length;i++)
        result[i]=array[i];
    calcrc();

    printf("\nChecksum is  : ");
    for (i=0;i<length;i++)
    {
        printf("%d",result[i]);
        if(result[i]!=0)
        flag=1;
     }
     if(flag)
     printf("\n  error in transmission\n");
     else
     printf("\n No error in transmission\n");



    getch();
}



void calcrc()
{
int pos=0,j,i;

while(pos<(length-degree))
{
j=0;

for(i=pos;i<=(pos+degree);i++,j++)
{
result[i]=result[i]^gen[j];

}
i=pos;
while(result[i++]==0)
pos++;
}
}
```

Note: first check the output by taking simple generator polynomial(say 11001) then take ccitt polynomial(x16 + x12 + x5 + 1) as generator polynomial.

Output:

Enter the data stream : 10011
The transmitted frame is : 1 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 1 0
Enter the stream for which CRC has to be checked : 100110010001001010010
Checksum : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
No Transmission Error

PART-B

1. Communication between two PC's using RS232.

Introduction

Data Communication refers to the exchange of digital information(data bits) between digital devices. The transfer of information is normally between two computers/between computer and their terminals.

There are two modes of transmission.

a.) Synchronous Transmission b) Asynchronous Transmission

In synchronous communication the receiver uses a clock which is synchronized to the transmitter clock. Synchronous transmission has the advantage that the timing information is accurately aligned to the received data, allowing operation at much higher data rates.

In Asynchronous communications is the method of communications most widely used for PC communication and is commonly used for e-mail applications, Internet access, and asynchronous PC-to-PC communications. Through asynchronous communications, data is transmitted one byte at a time with each byte containing one start bit, eight data bits, and one stop bit, thus yielding a total of ten bits. With asynchronous communications, there is a high amount of overhead because every byte sent contains two extra bits (the start and stop bits) and therefore a substantial loss of performance.

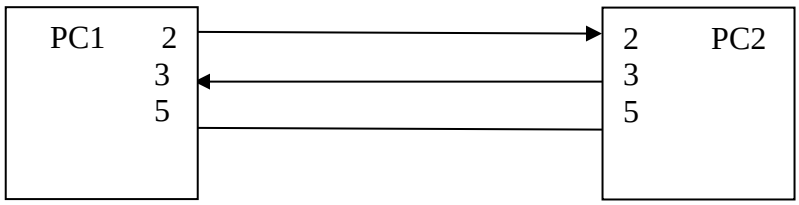
RS-232 Communication:

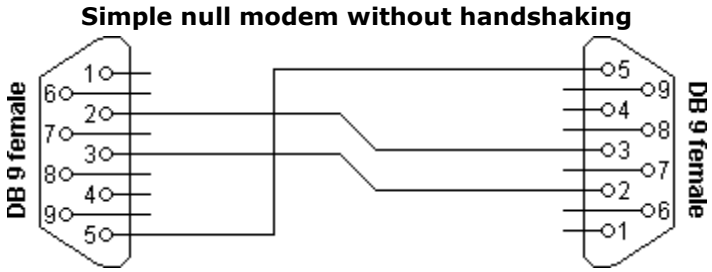
In telecommunications, **RS-232** (Recommended Standard 232) is a standard for serial binary single-ended data and control signals connecting between a *DTE* (Data Terminal Equipment) and a *DCE* (Data Circuit-terminating Equipment)/DTE. It is commonly used in computer serial ports. The standard defines the electrical characteristics and timing of signals, the meaning of signals, and the physical size and pinout of connectors. Communication as defined in the **RS232** standard is an asynchronous serial communication method. The word *serial* means, that the information is sent one bit at a time. *Asynchronous* tells us that the information is not sent in predefined time slots. Data transfer can start at any given time and it is the task of the receiver to detect when a message starts and ends. RS-232 Protocol identifies how fast data is transmitted between two ports. This transmission speed is defined as Baud Rate, roughly equivalent to the number of bits transmitted per second. Typically, the baud rate will vary between 1200 to 19200. Common baud rates are as follows: 1200, 2400, 4800, 9600, and 19200.

Null Modem:

Null modem is a communication method to connect two DTEs (computer, terminal, printer etc.) directly using an RS-232 serial cable. The RS-232 standard is asymmetrical as to the definitions of the two ends of the communications link so it assumes that one end is a DTE and the other is a DCE e.g. a modem. With a null modem connection the transmit and receive lines are **crosslinked**.

The figure shows the connection (called NULL MODEM connection) between two PC's.





Connector 1	Connector 2	Function	
2	3	Rx ←	Tx
3	2	Tx →	Rx
5	5	Signal ground	

Hyperterminal:

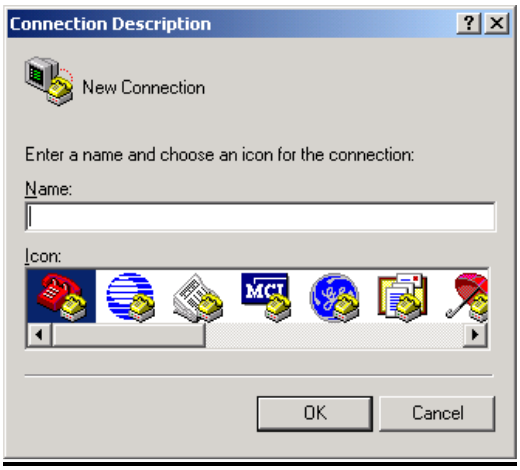
HyperTerminal (also known as HyperTerm) is a communications and terminal emulation program that comes with the Windows operating system, beginning with Windows 98.

Procedure:

1. In first terminal(first PC1) Go to

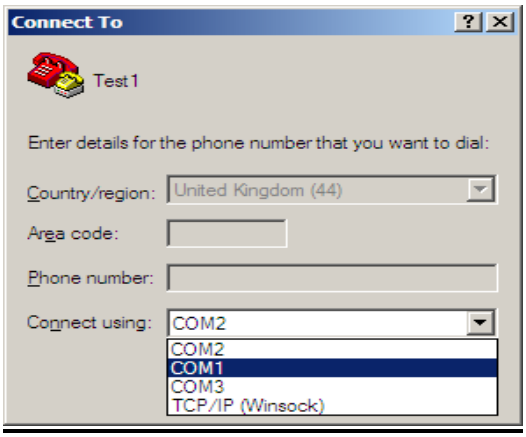
   START>PROGRAMS>ACCESSORIES>HYPERTERMINAL

2. The first window that will pop up is called *Connection Description*. This allows you to NAME the connection and assign an ICON to it. Here we have named it "RS-232 Troubleshooting" and selected one of the icons available from the window.

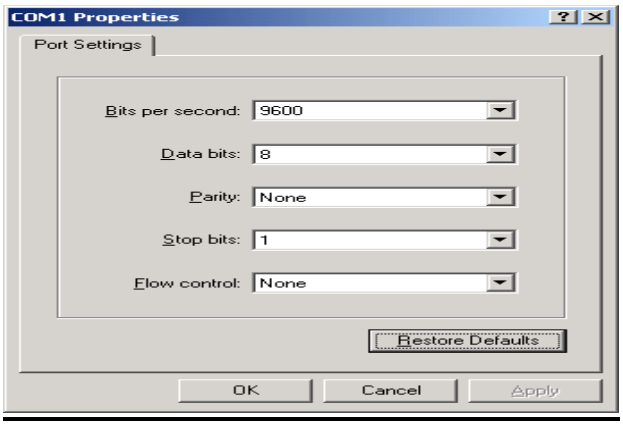


3. Type in a connection identifier name say “Test1” and click OK. The only restriction here is it cannot be a device name e.g. “COM1”.

4. You will then be presented with the following dialogue box: Select COM1



5. The next dialogue box will ask for the Port Communication Settings. Set Baud rate, number of data bits ,parity and stop bits..



6. Type the program for serial communication in other terminal (PC2). Execute the following program.

```
#include<stdio.h>
#include<conio.h>
#include<bios.h>
#define SETTINGS (_COM_9600 | _COM_CHR8 |_COM_NOPARITY|_COM_STOP1)
/* baud rate = 9600, 8 data bits, no parity bit, 1 stop bit */
void main(void)
{
unsigned int in,out,status;
int port=0;
clrscr();

cprintf("\n\rData Received:");
_bios_serialcom(_COM_INIT,port,SETTINGS);
for(;;)
{

if(kbhit()) /* if a keystroke is currently available */
{
in=getche(); /* get a character with echoing onto the screen */
if(in==27) /* if ESC */
break;
_bios_serialcom(_COM_SEND,port,in);
}

status=_bios_serialcom(_COM_STATUS,port,0);

if(status & 256) /* if data ready */
{
if((out=_bios_serialcom(_COM_RECEIVE,port,0) & 255)!=0)
```

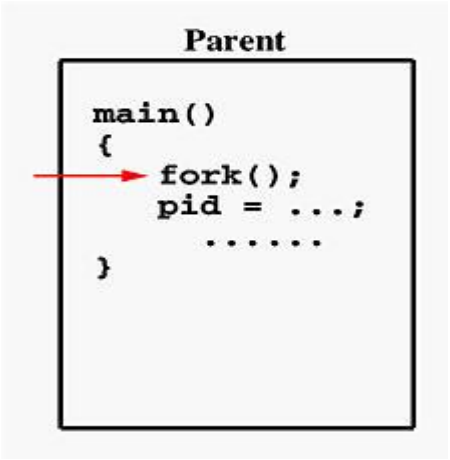
```
putch(out);
}
}
}
```

Output:
Enter the input from any of the PC's get the output in other PC.

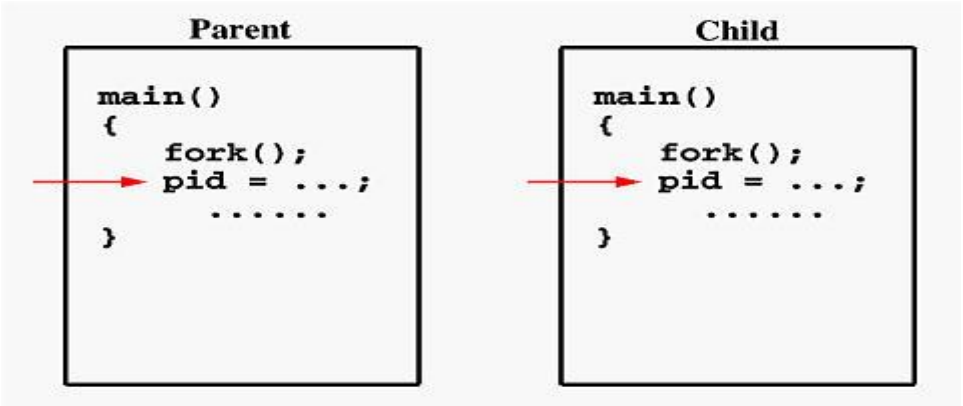
2. Inter Process Communication between Child and Parent

fork() system call: System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a *new* process, which becomes the *child* process of the caller. After a new child process is created, *both* processes will execute the next instructions following the *fork()* system call (defined in sys/types.h). Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of fork(): fork() returns a zero to the newly created child process. fork() returns a positive value, the *process ID* of the child process, to the parent Normally, the process ID is an integer. Moreover, a process can use function getpid() to retrieve the process ID assigned to this process.

- Before fork()

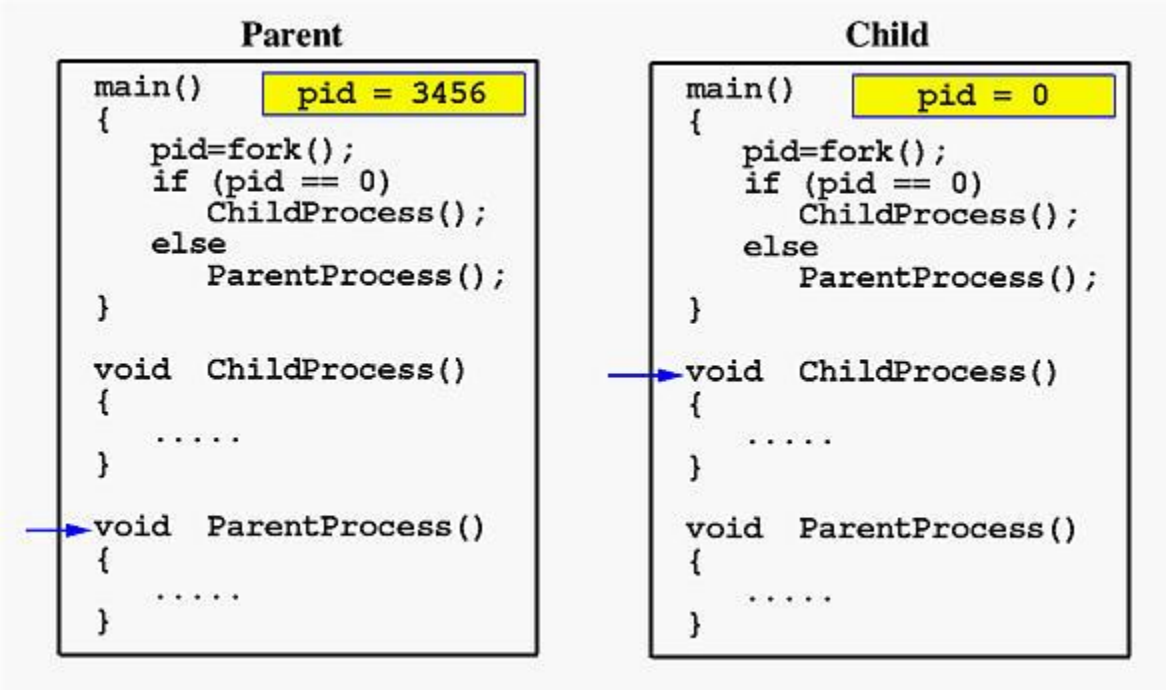


After fork()



Both processes start their execution right after the system call fork(). Since both processes have identical but separate address spaces, those variables initialized before the fork() call have the same values in both address spaces. Since every process has its own address space,

any modifications will be independent of the others. In other words, if the parent changes the value of its variable, the modification will only affect the variable in the parent process's address space. Other address spaces created by fork() calls will not be affected even though they have identical variable names. Now both programs (*i.e.*, the parent and child) will execute independent of each other starting at the next statement. In the parent, since pid is non-zero, it calls function ParentProcess(). On the other hand, the child has a zero pid and calls ChildProcess() as shown below:



pipe() System call:

To create a simple pipe with C, we make use of the pipe() system call. It takes a single argument, which is an array of two integers, and if successful, the array will contain two new file descriptors to be used for the pipeline The first integer in the array (element 0) is set up and opened for reading, while the second integer (element 1) is set up and opened for writing. If the parent wants to receive data from the child, it should close fd1 by using close(fd[1]); and the child should close fd0. If the parent wants to send data to the child, it should close fd0, and the child should close fd1. Since descriptors are shared between the parent and child, we should always be sure to close the end of pipe we aren't concerned with

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include<string.h>

int main(void)
{
      int     fd[2], nbytes;
      int pid;
      char    string[50];
      char    readbuffer[80];

      pipe(fd);

       pid=fork();

      if(pid == 0)
      {
        printf(" i am in child and pid variable=%d\n",pid);
        printf("enter the string to be passed to parent");
        gets(string);

        close(fd[0]);

         write(fd[1], string, (strlen(string)+1));

      }
      else
      {
            printf("i am in parent and pid variable=%d\n",pid);


            close(fd[1]);

            /* Read in a string from the pipe */
            nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
            printf("\nReceived string: %s , number of bytes=%d", readbuffer,nbytes);
      }

      return(0);
}
```

Output: I am in child and pid variable=0
Enter the string to be passed to parent hello how r u
I am in parent and pid variable=2367
Received string: hello how r u, number of bytes=14

3. Communicate between TWO Processes/PCs using simple socket function.

A socket is a piece of code which we write to enable communication between two separate entities, meaning computers in this context, or within the entity itself. Sockets can also be used for inter process communication on a single computer. Applications can create multiple sockets for communicating with each other. Sockets are bidirectional, meaning that either side of the connection is capable of both sending and receiving data.

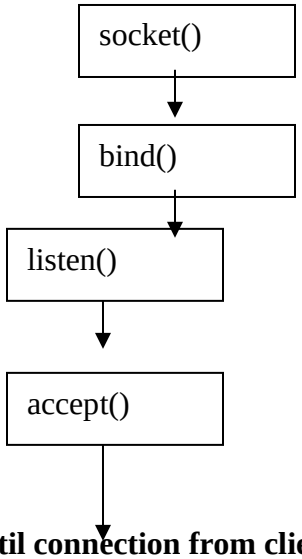
Most interprocess communication uses the client server model. These terms refer to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server, typically to make a request for information.

Notice that the client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established

A server application normally listens to a specific port waiting for connection requests from a client. When a connection request arrives, the client and the server establish a dedicated connection over which they can communicate. During the connection process, the client is assigned a local port number, and binds a *socket* to it. The client talks to the server by writing to the socket and gets information from the server by reading from it. Similarly, the server gets a new local port number (it needs a new port number so that it can continue to listen for connection requests on the original port). The server also binds a socket to its local port and communicates with the client by reading from and writing to it. The client and the server must agree on a protocol--that is, they must agree on the language of the information transferred back and forth through the socket.

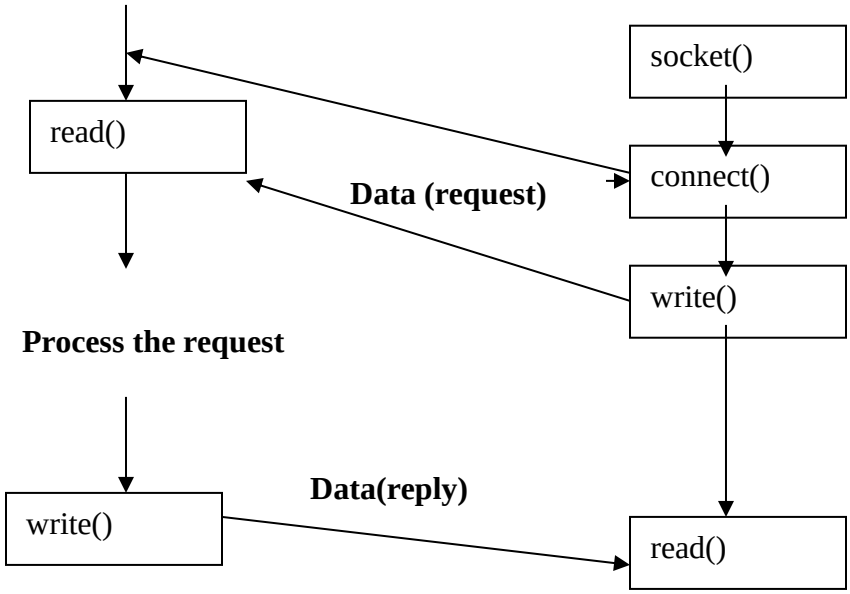
Fig shows a typical scenario that takes place for a connection-oriented transfer-first the server started, then sometimes later a client is started that connects the server.

SERVER



Blocks until connection from client.

CLIENT



Creating a Socket

The socket system call prototype is given below:

int socket(int family, int type, int protocol);

The **family** is either AF_UNIX or AF_INET. An AF_UNIX socket can only be used for interprocess communications on a single system, while an AF_INET socket can be used for communications between systems.

The **type** specifies the characteristics of communication on the socket. SOCK_STREAM creates a socket that will reliably deliver bytes in-order, but does not respect messages boundaries; SOCK_DGRAM creates a socket that does respect message boundaries, but does not guarantee to deliver data reliably, uniquely or in order. A SOCK_STREAM socket corresponds to TCP; a SOCK_DGRAM socket corresponds to UDP.

The **protocol** selects a protocol. Ordinarily this is 0, allowing the call to select a protocol. This is almost always the right thing to do, though in some special cases you may want to select the protocol yourself. Remember that this refers to the underlying network protocol: such well-known protocols as http, ftp, and ssh are all built on top of tcp, so tcp is the right choice for any of those protocols.

This system call returns a small integer value(ID of the socket created), similar to a file descriptor. This is known as Socked descriptor.

For example:

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

will create a socket that will use TCP to communicate. The return value (s) is a file descriptor for the socket.

Binding a Socket:

At this point created a socket, but name is not given to it. To give a name to the socket by using the bind system call:

int bind(int sockfd, struct sockaddr *myaddr, int addrlen);

where, sockfd is socket descriptor which is already created and to be binded.

The struct sockaddr is defined as follows:

```
struct sockaddr_in {
   sa_family_t    sin_family; /* address family: AF_INET */
   u_int16_t      sin_port;   /* port in network byte order */
   struct in_addr  sin_addr;  /* internet address */
};

/* Internet address. */
```

```
struct in_addr {
   u_int32_t      s_addr;     /* address in network byte order */
};
```

sin_family is always AF_INET; *sin_port* is the port number, and *sin_addr* is the IP address. Port numbers below 1024 are reserved - that means only processes with an effective user id of 0 (ie the root) can bind to those ports.

Listen:

Once the socket has been created and bound, the daemon needs to indicate that it is ready to listen to it. It is usually executed after both the socket and bind system calls, and immediately before the accept system call. It does this with the listen system call. Prototype of the system call is as follows:

int listen(int sockfd, int backlog);

sockfd is as explained earlier.
The backlog argument specifies how many connection requests can be queued by the system while it waits for the server to execute the accept system call. This argument is usually specified as 5, the maximum value currently allowed.

accept:

An actual connection from some client process is waited for by having the server execute the accept system call. Prototype of this call is as follows:

int accept(int sockfd, struct sockaddr *peer, int *addrlen);

accept takes the first connection request on the queue and creates another socket with the same properties as sockfd and returns the descriptor of new socket created per request. If there are no connection requests pending, this call blocks the caller until one arrives. **Peer** is the address of the connected peer process(the client). The addrlen is length of the sockaddr structure.

Connect:

A client connects to the socket using the connect call. First it creates a socket using the socket call, then it establishes a connection with server.

int connect(int scokfd, struct sockaddr *servaddr, int addrlen);

This call returns *a new socket*. This means the server can communicate with the client using the newly created (and unnamed) socket, while continuing to listen on the old one. At this point, the server normally forks a child process to handle the client, and goes back to its accept loop. The child doing the communication can either use standard read and write calls, or it can use send and recv. These calls work like read and write, except that you can also pass flags allowing for some options.

Sending Data

There are a variety of functions that may be used to send outgoing messages.
write() may be used in exactly the same way as it is used to write to files. This call may only be used with SOCK_STREAM type sockets.
int write(int fd, char *msg, int len);
fd is the socket descriptor. **msg** specifies the buffer holding the text of the message, and **len** specifies the length of the message.

Receiving Data

There are a variety of functions that may be used to receive incoming messages.

read() may be used in the exactly the same way as for reading from files. There are some complications with non-blocking reads. This call may only be used with SOCK_STREAM type sockets.
nt read(int fd, char *buff, int len)

fd is the socket descriptor. **buff** is the address of a buffer area. **len** is the size of the buffer.

Close:

The close system call is used to close a socket.

int cliose(int fd);

In order to use above explained functions, the following header files should be included:
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>

Program in C/ C++ for client server communication using TCP or IP sockets to make client send the name of the file and server to send back the contents of the requested file if present.

SERVER:

```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sdtio.h>
#include<errno.h>
#include<string.h>

int main()
{
    int sockfd,newsockfd,portno=1234,servlen ,n,fd;
    char buffer[256],c[2000];
    struct sockaddr_in serv_addr;

  sockfd=socket(AF_INET,SOCK_STREAM,0);
  bzero((char *)&serv_addr,sizeof(serv_addr));

     serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=INADDR_ANY;
    serv_addr.sin_port=htons(portno);

    bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr));

 listen(sockfd,5);

servlen=sizeof(serv_addr);
    printf("SERVER:Waiting for client....\n");
    newsockfd=accept(sockfd,(struct sockaddr *)&serv_addr,&servlen);
  prinf("ACCEPTED"):

 bzero(buffer,255);
  n=read(newsockfd,buffer,255);
   printf("SERVER: %s\n",buffer);


  fd=open(buffer,O_RDONLY,0);
   read(fd,c,255);
 write(newsockfd,c,255);

close(newsockfd);
```

```
close(sockfd);
close(fd);
return 0;
}
```

CLIENT:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>


int main()
{
   int sockfd,portno=1234,n;  //use same port as the server
   struct sockaddr_in serv_addr;
   struct hostent *server;
   char buffer[256],c[2000];



   sockfd=socket(AF_INET,SOCK_STREAM,0);


   bzero((struct sockaddr_in *)&serv_addr,sizeof(serv_addr));

   serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=INADDR_ANY;
   serv_addr.sin_port=htons(portno);


   connect(sockfd,&serv_addr,sizeof(serv_addr));
   perror("Connect");

   printf("CLIENT: Enter path with filename:\n");
   scanf("%s",&buffer);
   write(sockfd,buffer,255);

   bzero(c,255);
   n=read(sockfd,c,255);

   printf("CLIENT:Displaying contents of %s\n",c);
   close(sockfd);
   return 0;
}
```