U23ECP43-SIGNAL PROCESSING AND NETWORKS LABORATORY

LIST OF EXPERIMENTS

- **LIST OF EXPERIMENTS (SIGNAL PROCESSING USING MATLAB)**

1. Generation of elementary Discrete-Time sequences.
2. Linear and Circular convolutions.
3. Autocorrelation and Cross Correlation.
4. Frequency Analysis using DFT.
5. Design of FIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation.
6. Design of Butterworth and Chebyshev IIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operations.

- **LIST OF EXPERIMENTS (NETWORKING)**

7. Implementation of Error Detection/ Error Correction Techniques.
8. Implementation of Stop and Wait Protocol and sliding window.
9. Implementation of IP address configuration.
10. Data encryption and decryption using RSA (Rivest, Shamir and Adleman) algorithm.
11. Network Topology - Star, Bus, Ring.
12. Implementation of Link state routing algorithm.

EX. NO. :1	GENERATION OF SIGNALS
DATE:	

AIM :

To generate different signals using MATLAB 7.0.4

ALGORITHM :

1. Get the required values for number of samples n.
2. Specify X-axis coordinates level.
3. Specify Y-axis coordinate level according to the appropriate mathematical expression of different signal.
4. Plot the all the different signals.
5. Specify suitable X-axis and Y-axis label.
6. Specify suitable title for the signal.
7. If needed, Grid can be turned on.

PROGRAM:

```
clc;
clear;
n=input('enter the value of n:');
%Generation of unit step signal
        t=-n:1:n-1;
        k=[zeros(1,n),ones(1,n)];
        subplot(4,2,1);
        hold on;
        stem(t,k);
        hold off;
        xlabel('normalized frequency n');
        ylabel('amplitude x');
        title('unit step signal');
        grid on;
 %Generation of unit impulse signal
        t=-n:1:n-1;
        z=[zeros(1,n),ones(1),zeros(1,n-1)];
       subplot(4,2,2);
        hold on;
        stem(t,z);
        hold off;
        xlabel('normal n');
        ylabel('amp');
        title('unit impulse signal');
        grid on;
 %Generation of rising exponential signal
        n=[0:0.2:5];
        m=exp(n);
        subplot(4,2,3);
        hold on;
        stem(n,m);
        hold off;
        xlabel('frequency n');
        ylabel('amplitude x');
```

```
        title('rising exponential signal');
        grid on;

%Generation of decaying exponential signal
        p=exp(-n);
        subplot(4,2,4);
        hold on;
        stem(n,p);
        hold off;
        xlabel('normalized frequency n');
        ylabel('amplitude  x');
        title('decaying  exponential signal');
        grid on;
%Generation of the ramp signal
        x=0:0.5:10;
        y=x;
        subplot(4,2,5);
        hold on;
        stem(y);
        hold off;
        xlabel('normalized frequency n');
        ylabel('amplitude  x');
        title('ramp signal');
        grid on;
%Generation of the sine wave signal
        t=0:0.01:pi;
        y=sin(2* pi* t);
        subplot(4,2,6);
        hold on;
        plot(t,y);
        hold off;
        xlabel('normalized frequency n');
        ylabel('amplitude  x');
        title('sine wave signal');
        grid on;
%Generation of the cosine wave signal
        t=0:0.01:pi;
        y=cos(2* pi* t);
        subplot(4,2,7);
        hold on;
        plot(t,y);
        hold off;
        xlabel('normalized frequency n');
        ylabel('amplitude  x');
        title('cosine wave signal');
        grid on;
%Generation of the sawtooth waveform signal
        a=input('enter the amplitude of sawtooth waveform:');
        t=0:0.2:10;
        y=a*sawtooth(t);
        subplot(4,2,8);
        hold on;
        stem(t,y);
```

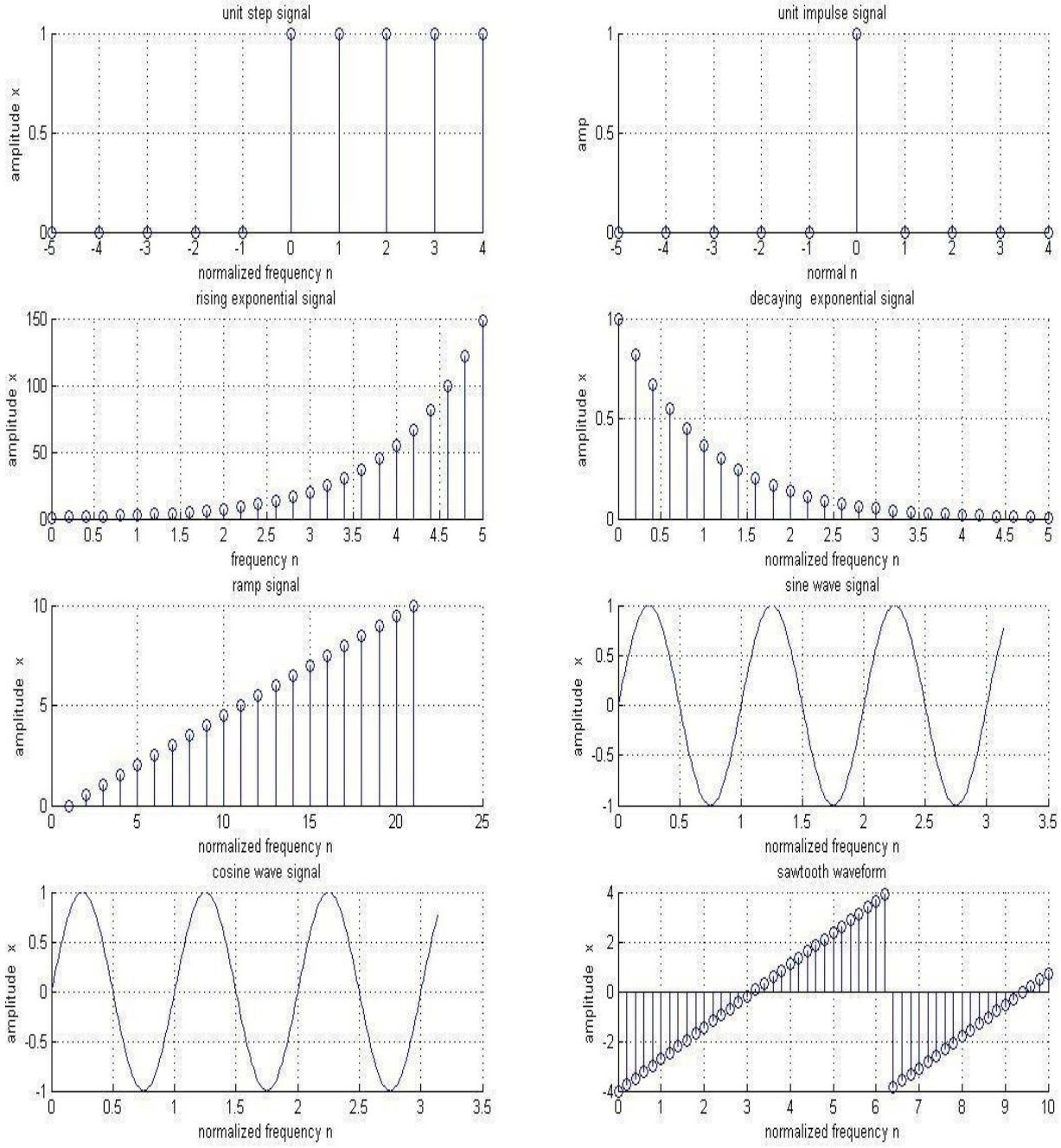
```
hold off;
xlabel('normalized frequency n');
ylabel('amplitude  x');
title('sawtooth waveform');
grid on;
```

INPUT:

Enter the value of n : 5

Enter the amplitude of sawtooth waveform : 4

OUTPUT:



RESULT :

Thus the different types of signals were generated successfully using MATLAB 7.0.4.

EX. NO. :2a	LINEAR CONVOLUTION OF TWO SEQUENCES
DATE:	

AIM:

To perform Linear Convolution of two given sequences using MATLAB 7.0.4

ALGORITHM :

1. Get the values for the Input sequence x(n).
2. Get the values for the Impulse sequence h(n).
3. Perform the Linear and Circular convolution of x(n) and h(n).
4. Plot the Input sequence x(n) and Impulse sequence h(n).
5. Plot the Linear convolution sequences X(k).
6. Specify suitable X-axis and Y-axis label.
7. Specify suitable title for the signal.
8. If needed, Grid can be turned on.

PROGRAM:

```
clc;
clear;
x=input('ENTER THE INPUT SEQUENCE x(n):');
h=input('ENTER THE IMPULSE SEQUENCE h(n):');
y=conv(x,h);
subplot(3,1,1);
stem(x);
xlabel ('n');
ylabel ('x(n)');
title('INPUT SEQUENCE x(n)');
grid on;
subplot(3,1,2);
stem(h);
xlabel ('n');
ylabel ('h(n)');
title('IMPULSE SEQUENCE h(n)');
grid on;
subplot(3,1,3);
stem(y);
xlabel ('n');
ylabel ('y(n)');
title('LINEAR CONVOLUTION OFx(n)and h(n)');
grid on;
disp(x);
disp(h);
disp(y);
```

INPUT:

ENTER THE INPUT SEQUENCE x(n):[1 2 3 4]

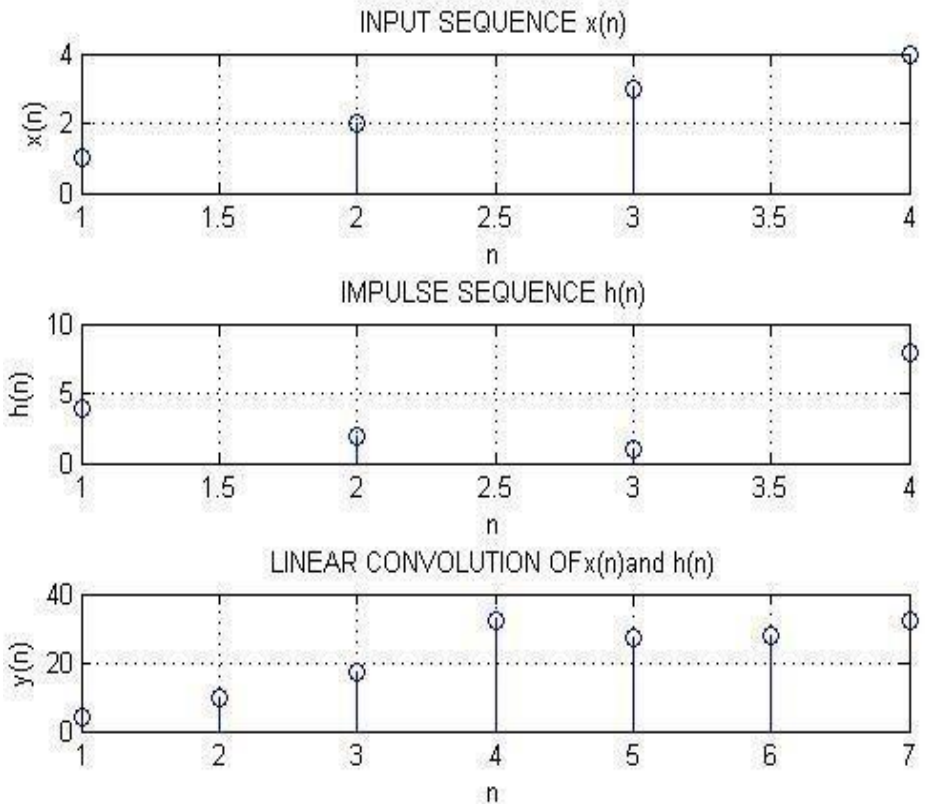
ENTER THE IMPULSE SEQUENCE h(n):[4 2 1 8]

1 2 3 4

4 2 1 8

4 10 17 32 27 28 32

OUTPUT:



RESULT:

Thus the Linear Convolution of two given sequences was performed successfully using MATLAB 7.0.4

EX. NO. :2b	CIRCULAR CONVOLUTION OF TWO SEQUENCES
DATE:	

AIM

To perform Circular Convolution of two given sequences using MATL 7.0.4

ALGORITHM :

1. Get the values for the Input sequence x(n).
2. Get the values for the Impulse sequence h(n).
3. Perform the Linear and Circular convolution of x(n) and h(n).
4. Plot the Input sequence x(n) and Impulse sequence h(n).
5. Plot the Circular convolution sequences X(k).
6. Specify suitable X-axis and Y-axis label.
7. Specify suitable title for the signal.
8. If needed, Grid can be turned on.

PROGRAM:

```
clc;
 clear all;
x=input('enter the input sequence x(n):');
h=input('enter the impulse sequence h(n):');
x1=fft(x);
h1=fft(h);
N=length(x);
for n=1:N
y1(n)=x1(n)*h1(n);
end
y=ifft(y1);
subplot(3,1,1);
stem(x);
xlabel('normalized frequency n');
ylabel('frequency x(n)');
title('input sequence x(n)');
grid on;
subplot(3,1,2);
stem(h);
xlabel('normalized frequency n');
ylabel('frequency x(n)');
title('impulse sequence h(n)');
grid on;
subplot(3,1,3);
stem(y);
xlabel('normalized frequency n');
ylabel('frequncy x(n)');
title('circular convolution of x(n) and h(n)');
grid on;
disp(x);
disp(h);
disp(y);
```

INPUT:

Enter the input sequence x(n):[1 2 3 4]

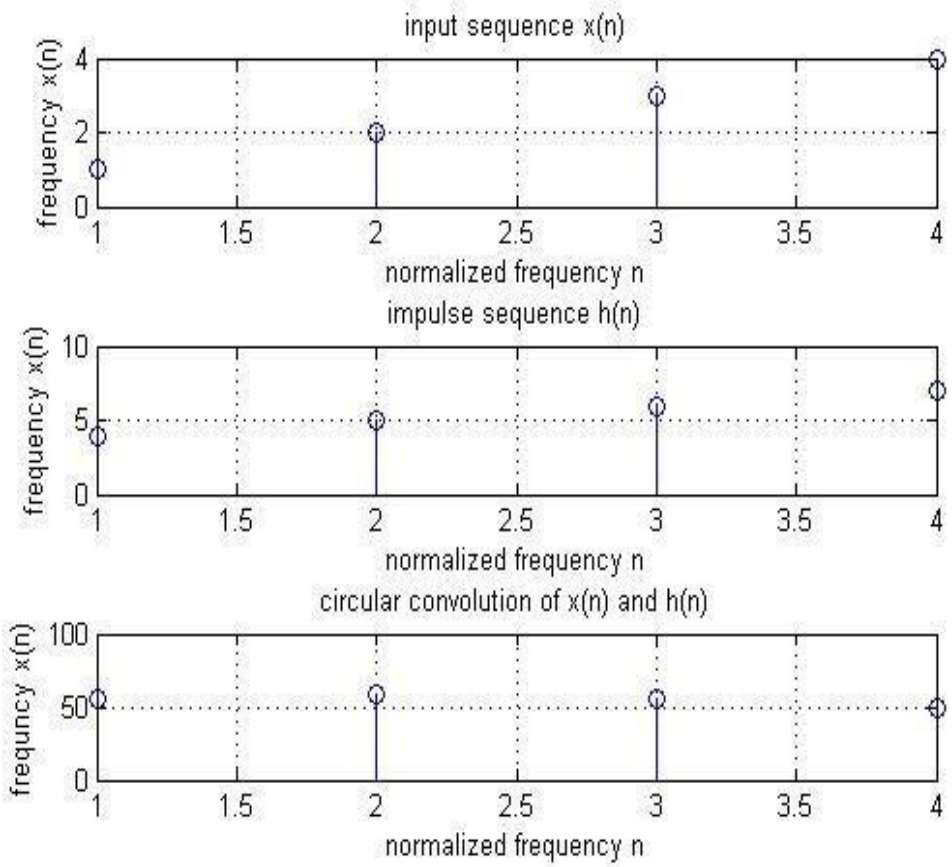
Enter the impulse sequence h(n):[4 5 6 7]

1 2 3 4

4 5 6 7

56 58 56 50

OUTPUT:



RESULT:

Thus the Circular convolution of two given sequences was performed successfully using MATLAB 7.0.4

EX. NO. :3a	AUTO CORRELATION OF TWO SEQUENCES
DATE:	

AIM:

To compute auto correlation between two sequences using MATLAB 7.0.4

Algorithm:

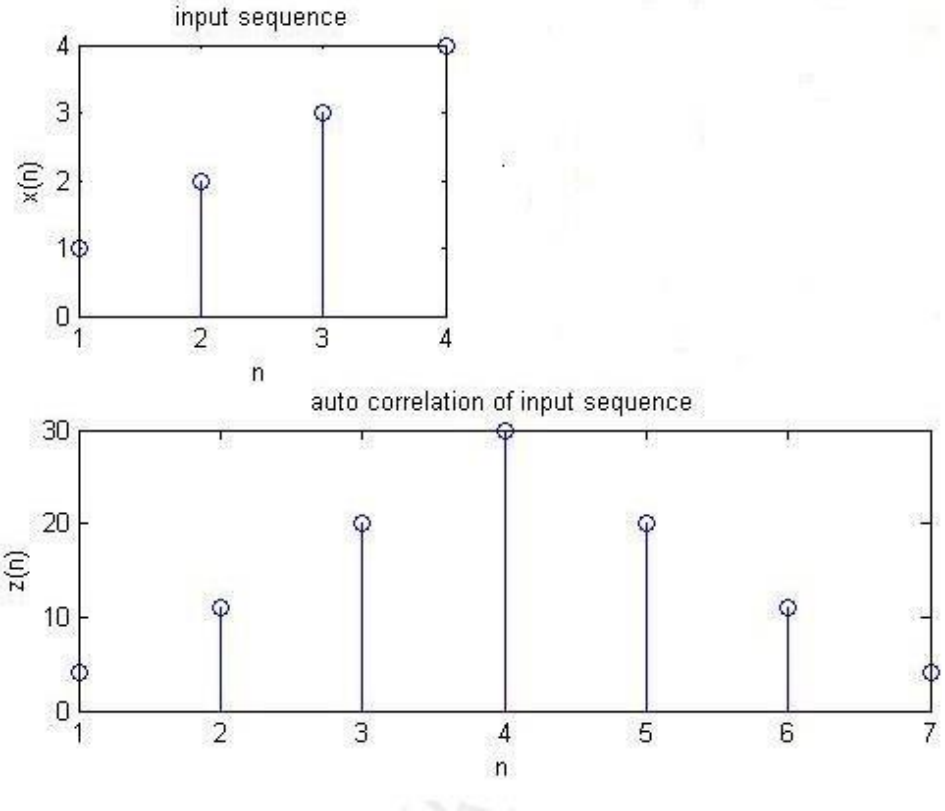
1. Give input sequence x[n].
2. Give impulse response sequence h (n)
3. Find auto correlation using the matlab command xcorr.
4. Plot x[n], h[n], z[n]. Step
5. Display the results

PROGRAM:

```
clc;
 close all;
clear all;
% two input sequences
x=input('enter input sequence')
subplot(3,2,1);
stem(x);
xlabel('n');
ylabel('x(n)');
title('input sequence');
% auto correlation of input sequence
z=xcorr(x,x);
subplot(3,2,2);
stem(z);
xlabel('n');
ylabel('z(n)');
 title('auto correlation of input sequence');
 disp(x);
 disp(z);
```

OUTPUT:

```
enter input sequence
        [1 2 3  4]

    x = 2

1                3       4


1       2        3       4

4.0000    11.0000    20.0000    30.0000    20.0000    11.0000     4.0000
```



RESULT:

Thus the Auto correlation of two given sequences was performed successfully using MATLAB 7.0.4

EX. NO. :3b	CROSS CORRELATION OF TWO SEQUENCES
DATE:	

AIM:

To compute cross correlation between two sequences using MATLAB 7.0.4

Algorithm:

1. Give input sequence x[n].
2. Give impulse response sequence h (n)
3. Find cross correlation using the matlab command xcorr.
4. Plot x[n], h[n], z[n]. Step
5. Display the results

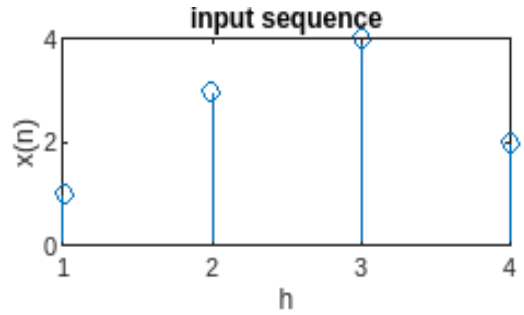
PROGRAM:

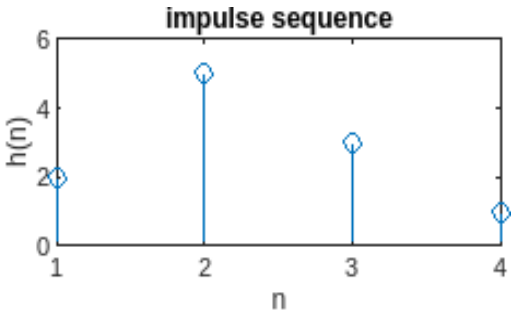
```
clc;
close all;
clear all;
x=input('enter the input sequence x(n)') ;
h=input('enter the impulse sequence h(n)');
subplot(3,2,1);
stem(x);
xlabel('h');
ylabel('x(n)');
title('input sequence');
subplot(3,2,2);
stem(h);
xlabel('n');
ylabel('h(n)');
title('impulse sequence');
% cross correlation of input sequence
z=xcorr(x,h);
subplot(3,2,3);
stem(z);
xlabel('n');
ylabel('z(n)');
 title('cross correlation of two sequence');
 disp(x);
 disp(h);
 disp(z);
```

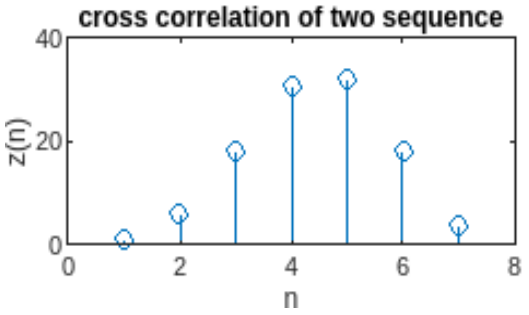
INPUT:

```
  enter the input sequence
x(n) [1 3 4 2]
enter the impulse sequence h(n)
  [2 5 3  1]
     1     3     4     2

     2     5     3     1

  1.0000    6.0000   18.0000   31.0000   32.0000   18.0000    4.0000
```

OUTPUT:







RESULT:

Thus the Cross correlation of two given sequences was performed successfully using MATLAB 7.0.4

EX. NO:4a	SPECTRUM ANALYSIS USING DFT
DATE:	

AIM:

To analyze the amplitude and phase response of the DFT and IDFT signal using MATLAB.

ALGORITHM:

1. Start the program.
2. Get the two input sequence x (n) and type of the DFT.
3. Compute the Discrete Fourier Transform (DFT) of the given sequence.
4. Find out the magnitude and phase response of the given sequence.
5. Execute the program, display the result and verify it.
6. Plot the output graph for magnitude and phase.
7. Compute the Inverse Discrete Fourier Transform (IDFT) of the given sequences.
8. Display the results verify it and plot the result.
9. Stop the process.

PROGRAM

```
% Program for Discrete Fourier Transform
clc;
clear all;
close all;
%x=[1 2 3 4 5 6 7 8];
%N=[8];
x=input('enter the input x=');
N=input('enter the order of dft N=');
k=0:1:N-1;
disp('Input Sequence x=');
disp(x);
 XK=fft(x,N);
disp('XK=');
disp(XK);
R=real(XK);
disp('Real part=');
disp(R);
I=imag(XK);
disp('Imaginary part=');
disp(I);
```

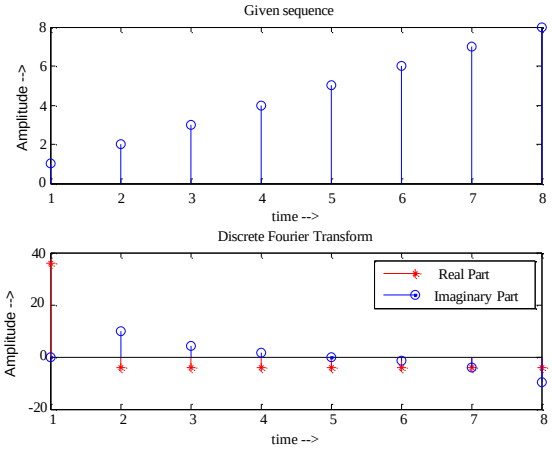
```
 mag=abs(XK);
disp('Magnitude=');
disp(mag);
ang=angle(XK);
disp('Angle=');
disp(ang);
xn=ifft(XK,N);
disp('IDFT xn=');
disp(xn);

figure(1);
subplot(2,1,1);
stem(x);
xlabel('time -->');
ylabel('Amplitude -->');
title('Given sequence');

subplot(2,1,2);
stem(R,'r*');
hold on;
stem(I);
legend('Real Part','Imaginary Part');
xlabel('time -->');
ylabel('Amplitude -->');
title('Discrete Fourier Transform')

figure(3);
subplot(2,2,1);
stem(k,mag);
xlabel('time -->');
ylabel('x(k) --->');
title('Magnitude response');

subplot(2,2,2);
stem(k,ang);
xlabel('time -->');
ylabel('ang x(k) --->');
title('Phase response');

subplot(2,2,3);
stem(k,xn);
xlabel('time -->');
ylabel('x(n)');
title('Inverse Discrete Fourier transform');
 %Spectrum analysis
Fs=[10];
xdft = XK(1:N/2+1);
psdx = (1/(Fs*N)) * abs(xdft).^2;
psdx(2:end-1) = 2*psdx(2:end-1);
```

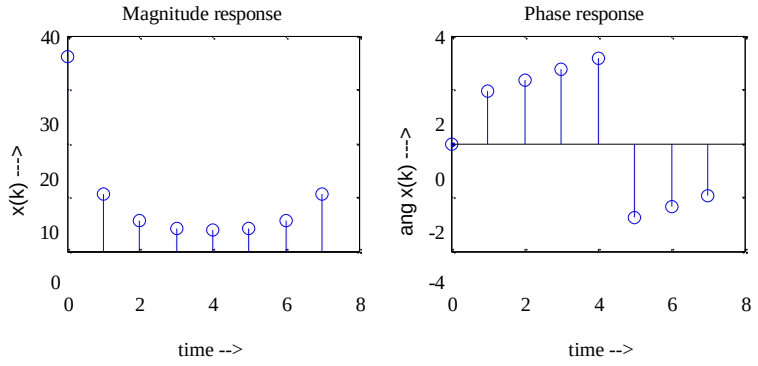
```
freq = 0:Fs/N:Fs/2;
subplot(2,2,4);
plot(freq,psdx);
grid on;
xlabel('hz -->');
ylabel('db');
title('Spectrum analysis DFT');
disp('Spectrum analysis DFT=');
disp(psdx);
```

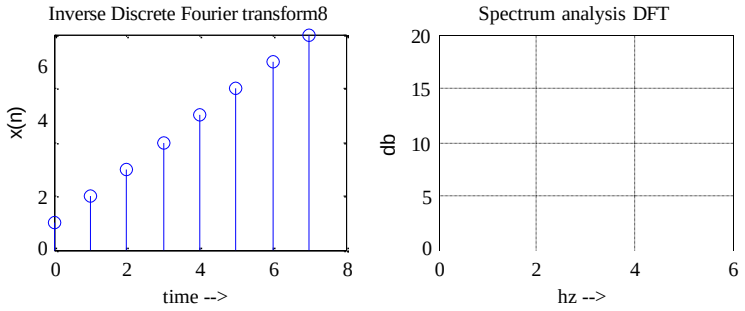
OUTPUT:

```
enter the input x=[1 2 3 4 5 6 7 8]
enter the order of dft N=[8]
 Input Sequence x=1     2     3     4     5     6     7     8

 XK=
  36.0000 + 0.0000i
  -4.0000 - 9.6569i
  -4.0000 - 4.0000i
  -4.0000 - 1.6569i
  -4.0000 + 0.0000i
  -4.0000 + 1.6569i
  -4.0000 + 4.0000i
  -4.0000 + 9.6569i

 Real part=   36    -4    -4    -4    -4    -4    -4    -4

 Imaginary part=      0    9.6569    4.0000    1.6569         0  -1.6569   -4.0000   -9.6569

 Magnitude= 36.0000   10.4525    5.6569    4.3296    4.0000    4.3296    5.6569  10.4525

 Angle= 0    1.9635    2.3562    2.7489    3.1416  -2.7489   -2.3562   -1.9635

 IDFT xn= [1     2     3     4     5     6     7     8]
 Spectrum analysis DFT= 16.2000     2.7314     0.8000     0.4686     0.2000
```

MODEL GRAPH







RESULT:

Thus a MATLAB program for Spectrum analysis of the DFT and IDFT signal was written, executed and the graphs were plotted.

EX. NO:4b	FAST FOURIER TRANSFORM AND INVERSE FAST TRANSFORM (FFT & IFFT)
DATE:	

AIM:

To write a program to find out the Fast Fourier transform and Inverse Fast Fourier transform of the sequence using MATLAB and display the magnitude and phase response.

ALGORITHM:

1. Start the program.
2. Get the two input sequence x (n) and type of the FFT.
3. Compute the fast Fourier transform (FFT) of the given sequence.
4. Find out the magnitude and phase response of the given sequence.
5. Execute the program, display the result and verify it.
6. Plot the output graph for magnitude and phase.
7. Compute the inverse fast Fourier transform (FFT) of the given sequences.
8. Display the results verify it and plot the result.
9. Stop the process.

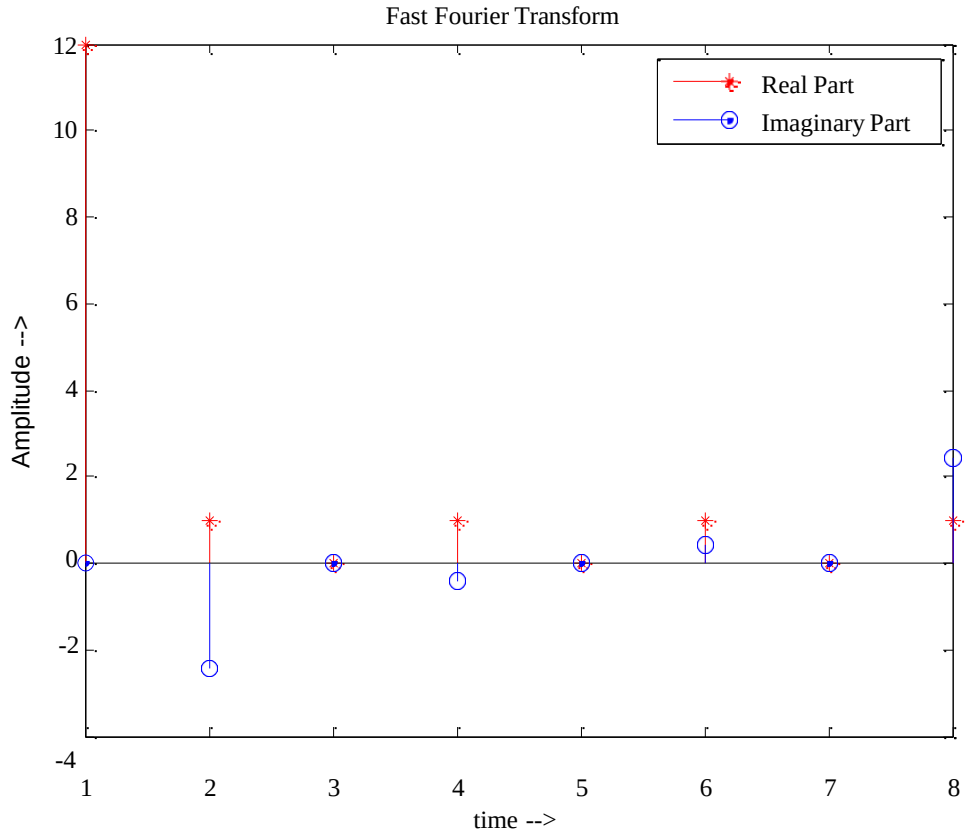
```
% Program for FFT & IFFT
clc;
clear all;
close all;
%x=[2 2 2 2 1 1 1 1];
%N=[8];
x=input('enter the input x=');
N=input('enter the order of fft N=');
k=0:1:N-1;

disp('Input Sequence x=');
disp(x);

XK=fft(x,N);
disp('XK=');
disp(XK);

R=real(XK);
disp('Real part=');
disp(R);

I=imag(XK);
disp('Imaginary part=');
disp(I);
```

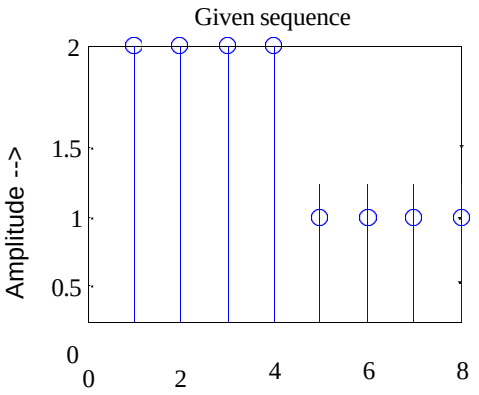
```
mag=abs(XK);
disp('Mag=');
disp(mag);
ang=angle(XK);
disp('angle=');
disp(ang);

xn=ifft(XK,N);
disp('IFFT xn=');
disp(xn);

figure(1);
stem(R,'r*');
hold on;
stem(I);
legend('Real Part','Imaginary Part');
xlabel('time -->');
ylabel('Amplitude -->');
title('Fast Fourier Transform')

figure(2);
subplot(2,2,1);
stem(x);
xlabel('time -->');
ylabel('Amplitude  -->');
title('Given  sequence');
subplot(2,2,2);
stem(k,mag);
xlabel('time -->');
ylabel('x(k) --->');
title('Magnitude response');

subplot(2,2,3);
stem(k,ang);
xlabel('time -->');
ylabel('ang x(k) --->');
title('Phase response');

subplot(2,2,4);
stem(k,xn);
xlabel('time -->');
ylabel('x(n)');
title('Inverse Fast Fourier transform');
```

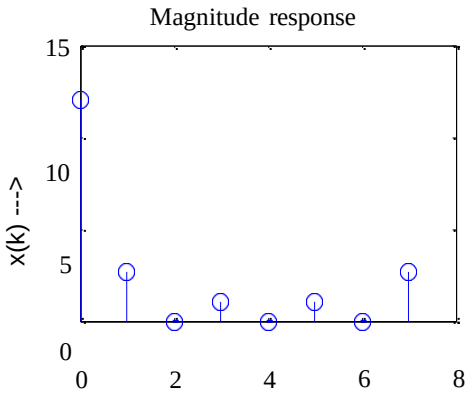
OUTPUT:

```
enter the input x=[2 2 2 2 1 1 1 1]
enter the order of fft N=[8]
Input Sequence x=    2     2     2     2     1     1     1     1
XK=
  12.0000 + 0.0000i
   1.0000 + 2.4142i
   0.0000 + 0.0000i
   1.0000 + 0.4142i
   0.0000 + 0.0000i
   1.0000 - 0.4142i
   0.0000 + 0.0000i
   1.0000 - 2.4142i

Real part=    12     1     0     1     0     1     0     1

Imaginary part=         0   -2.4142         0   -0.4142         0    0.4142         0    2.4142

Mag=   12.0000     2.6131         0    1.0824         0    1.0824         0    2.6131

Angle=        0   -1.1781         0   -0.3927         0    0.3927         0    1.1781

IFFT xn=    2     2     2     2     1     1     1     1
```

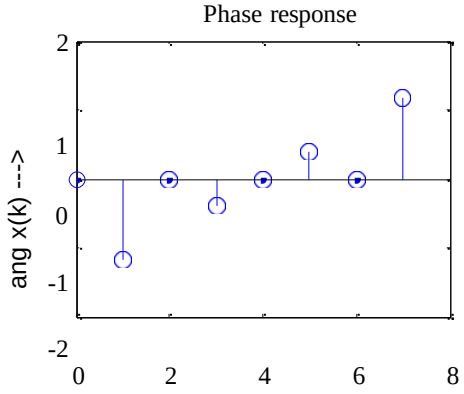
MODEL GRAPH:

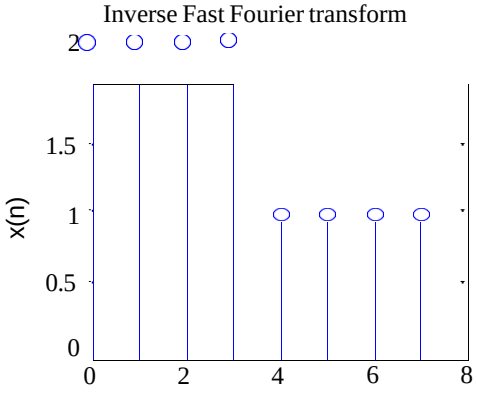


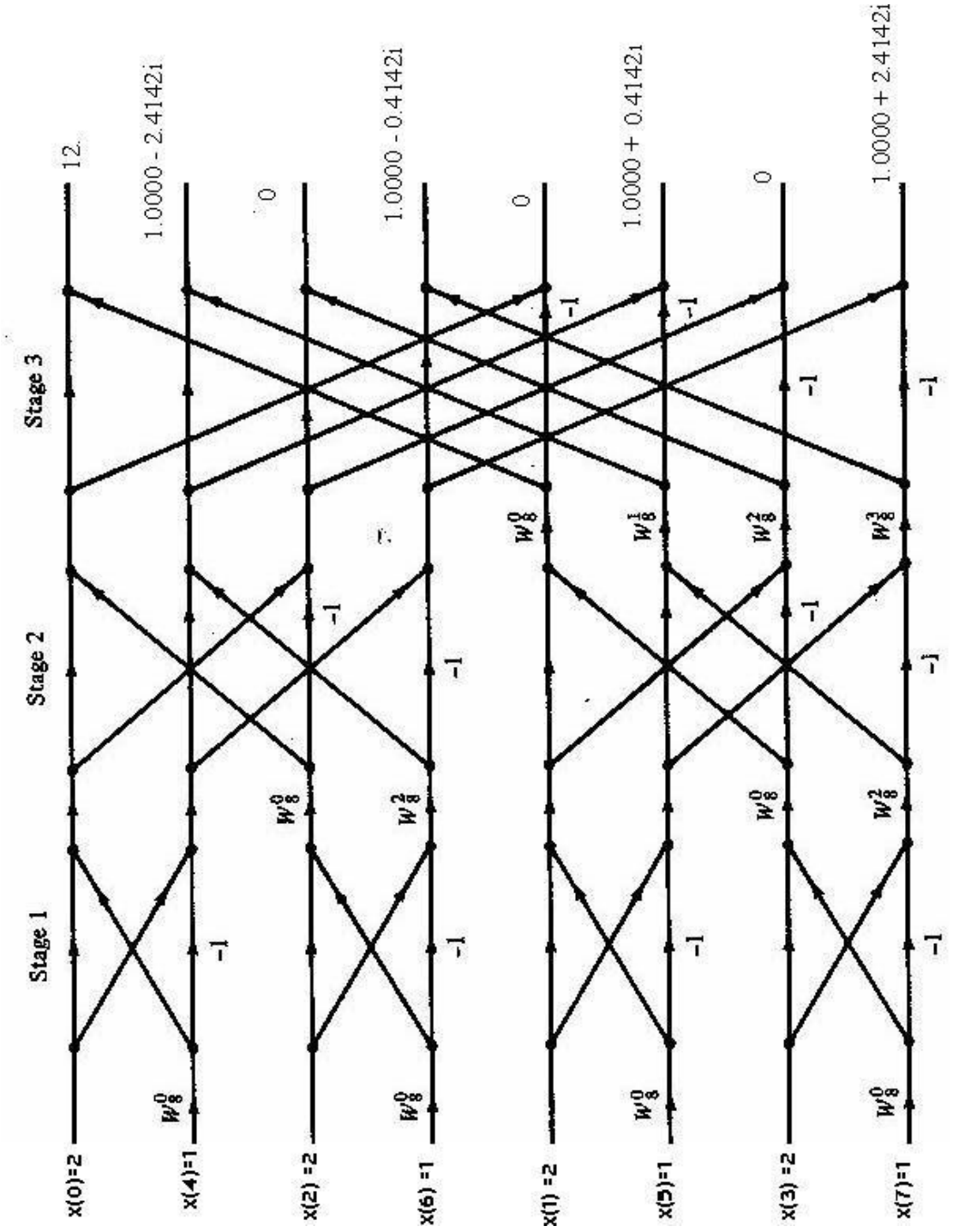




time --> time -->






Stage 1
Stage 2
Stage 3
x(0)=2
x(4)=1
x(2) =2
x(6) =1
x(1) =2
x(5)=1
x(3) =2
x(7)=1
12.
1.0000 - 2.4142i
0
1.0000 - 0.4142i
0
1.0000 + 0.4142i
0
1.0000 + 2.4142i
-1

RESULT:

Thus a program to find out the fast Fourier (FFT) and inverse fast Fourier transform (IFFT) of the sequence using MATLAB was written and executed.

EX. NO:5	FIR FILTER DESIGN
DATE:	

AIM:

To write a program to design the FIR low pass, high pass, band pass and band stop filters and obtain the frequency response of the filter using MATLAB.

ALGORITHM:

1. Start the program.
2. Get the cutoff frequency and order of the filter as input...
3. Find the ideal impulse response of the filter hd (n).
4. Select a window function w (n).
5. Find out the filter coefficients h (n) by multiplying hd (n) and w (n).
6. Compute the frequency response of the filter.
7. Repeat the process for high pass, band pass and band stop filters.
8. Execute the program, display the result and verify it.
9. Plot the output graph for hd(n),w(n),h(n)and frequency response of LPF,HPF,BPF and BSF.
10. Stop the process.

PROGRAM:

```
%  Ideal LowPass filter computation
%  hd = ideal impulse response between 0 to M-1
%  wc = cutoff frequency in radians
%  M = length of the ideal filter
%  Save as ideal_lp


% Function:
function [hd]=ideal_lp(wc,M)
alpha=(M-1)/2;
n=0:1:M-1;
N=n-alpha+eps;  % add smallest number to avoid divide by zero
hd=sin(wc*N)./(pi*N);


Program:
%PROGRAM FOR LOW PASS FILTER
clc;
clear all;
close all;
M=7;
wc=1.2;
n=0:1:M-1;
hd=ideal_lp(wc,M);
w=hamming(M);
h=hd.*w';
disp('Filter coefficients for LPF=');

[M1,P]=freqz(h,1,2000);
mag=20*log10(abs(M1));

subplot(2,2,1);
stem(n,hd);
xlabel('n--->');
ylabel('hd(n)--->');
title('Ideal Impulse Response');
disp('hd=');
disp(hd);
subplot(2,2,2);
stem(n,w);
xlabel('n--->');
ylabel('w(n)--->');
title('Hamming Window');
disp('w=');
disp(w);
subplot(2,2,3);
stem(n,h);
xlabel('n--->');
```

```
ylabel('h(n)--->');
title('Actual Impulse Response');
disp('h=');
disp(h);
subplot(2,2,4);
plot(P/pi, mag);
grid on;
xlabel('Normalized frequency--->');
ylabel('Magnitude in db--->');
title('Frequency Response ');
```

OUTPUT:

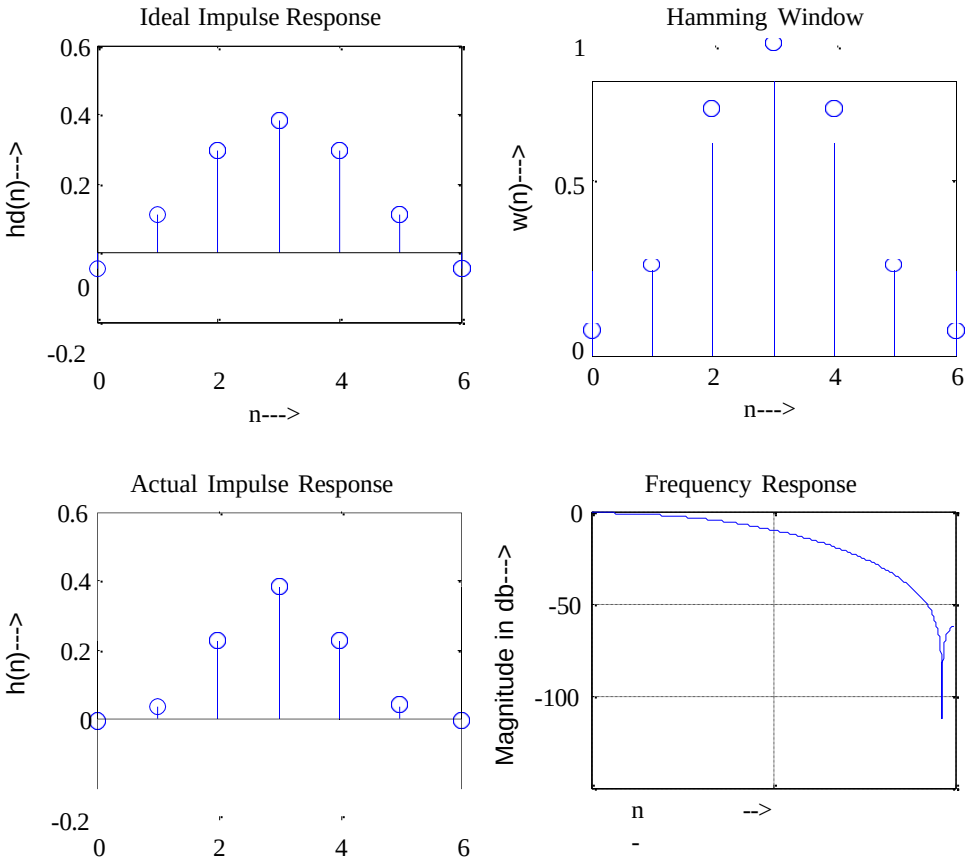
Filter coefficients for LPF=

hd= -0.0470 0.1075 0.2967 0.3820 0.2967 0.1075 -0.0470

w = 0.0800 0.3100 0.7700 1.0000 0.7700 0.3100 0.0800

h = -0.0038 0.0333 0.2284 0.3820 0.2284 0.0333 -0.0038

LOW PASS FILTER



```
%PROGRAM FOR HIGH PASS FILTER
        clc;
        clear all;
        close all;
        M=11;
        wc=1.2;
        n=0:1:M-1;
        hd=ideal_lp(pi,M)-ideal_lp(wc,M);
        w=hann(M);
        h=hd.*w';
        disp('Filter coefficients for HPF=');

        [M1,P]=freqz(h,1,2000);
        mag=20*log10(abs(M1));

        subplot(2,2,1);
        stem(n,hd);
        xlabel('n--->');
        ylabel('hd(n)--->');
        title('Ideal Impulse Response');
        disp('hd=');
        disp(hd);

        subplot(2,2,2);
        stem(n,w);
        xlabel('n--->');
        ylabel('w(n)--->');
        title('Hanning Window');
        disp('w=');
        disp(w);

        subplot(2,2,3);
        stem(n,h);
        xlabel('n--->');
        ylabel('h(n)--->');
        title('Actual Impulse Response');
        disp('h=');
        disp(h);

        subplot(2,2,4);
        plot(P/pi, mag);
        grid on;
        xlabel('Normalized frequency--->');
        ylabel('Magnitude in db--->');
        title('Frequency Response ');
```

OUTPUT

Filter coefficients for HPF=

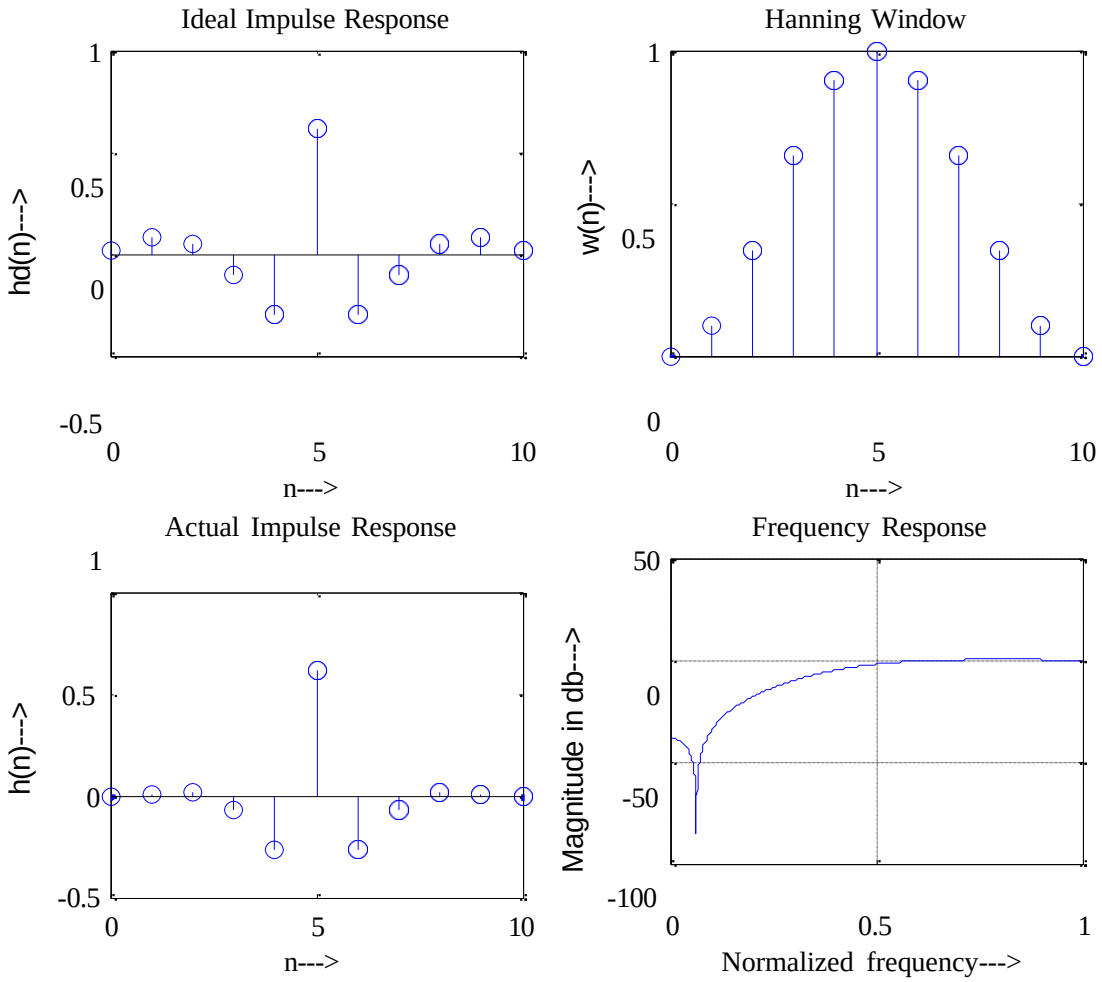
hd=0.0178 0.0793 0.0470 -0.1075 -0.2967 0.6180 -0.2967 -0.1075 0.0470 0.0793 0.0178

w = 0 0.0955 0.3455 0.6545 0.9045 1.0000 0.9045 0.6545 0.3455 0.0955 0

h = 0 0.0076 0.0162 -0.0704 -0.2683 0.6180 -0.2683 -0.0704 0.0162 0.0076 0

MODEL GRAPH:

HIGH PASS FILTER



```
%PROGRAM FOR BAND PASS FILTER
    clc;
    clear all;
    close all;
    M=11;
    wc1=1.2;
    wc2=2;
    n=0:1:M-1;
    hd=ideal_lp(wc2,M)-ideal_lp(wc1,M);
    w=hann(M);
    h=hd.*w';
    disp('Filter coefficients for BPF=');

    [M1,P]=freqz(h,1,2000);
    mag=20*log10(abs(M1));

    subplot(2,2,1);
    stem(n,hd);
    xlabel('n--->');
    ylabel('hd(n)--->');
    title('Ideal Impulse Response');
    disp('hd=');
    disp(hd);

    subplot(2,2,2);
    stem(n,w);
    xlabel('n--->');
    ylabel('w(n)--->');
    title('Hanning Window');
    disp('w=');
    disp(w);

    subplot(2,2,3);
    stem(n,h);
    xlabel('n--->');
    ylabel('h(n)--->');
    title('Actual Impulse Response');
    disp('h=');
    disp(h);

    subplot(2,2,4);
    plot(P/pi, mag);
    grid on;
    xlabel('Normalized frequency--->');
    ylabel('Magnitude in db--->');
    title('Frequency Response ');
```

OUTPUT:

Filter coefficients for BPF=

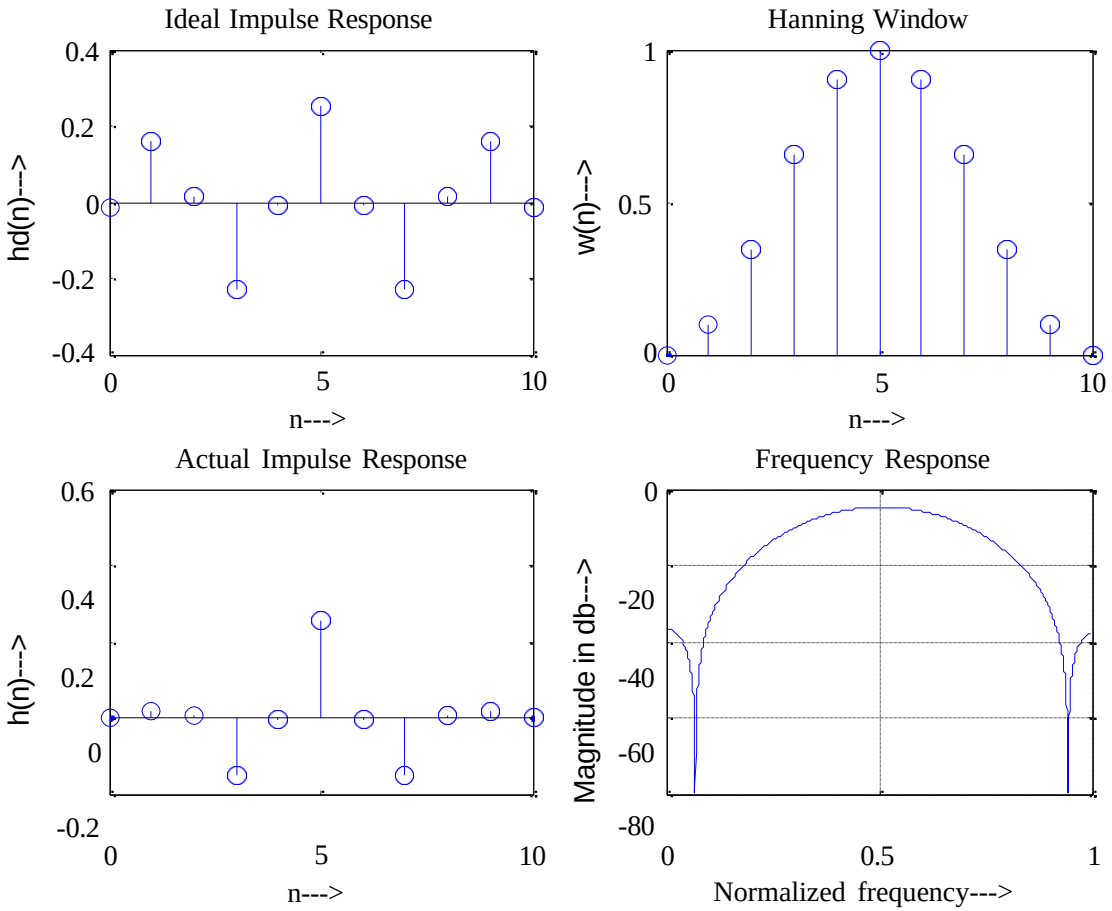
hd=-0.0168 0.1580 0.0173 -0.2280 -0.0072 0.2546 -0.0072 -0.2280 0.0173 0.1580 -0.0168

w= 0 0.0955 0.3455 0.6545 0.9045 1.0000 0.9045 0.6545 0.3455 0.0955 0

h= 0 0.0151 0.0060 -0.1492 -0.0065 0.2546 -0.0065 -0.1492 0.0060 0.0151 0

MODEL GRAPH:

BANDPASS FILTER



```
%PROGRAM FOR BAND STOP FILTER
    clc;
    clear all;
    close all;
    M=7;
    wc1=1.2;
    wc2=2;
    n=0:1:M-1;
    hd=ideal_lp(pi,M)-ideal_lp(wc2,M)+ideal_lp(wc1,M);
    w=rectwin(M);
    h=hd.*w';
    disp('Filter coefficients for BSF=');

    [M1,P]=freqz(h,1,2000);
    mag=20*log10(abs(M1));

    subplot(2,2,1);
    stem(n,hd);
    xlabel('n--->');
    ylabel('hd(n)--->');
    title('Ideal Impulse Response');
    disp('hd=');
    disp(hd);

    subplot(2,2,2);
    stem(n,w);
    xlabel('n--->');
    ylabel('w(n)--->');
    title('Rectangular Window');
    disp('w=');
    disp(w);

    subplot(2,2,3);
    stem(n,h);
    xlabel('n--->');
    ylabel('h(n)--->');
    title('Actual Impulse Response');
    disp('h=');
    disp(h);

    subplot(2,2,4);
    plot(P/pi, mag);
    grid on;
    xlabel('Normalized frequency--->');
    ylabel('Magnitude in db--->');
    title('Frequency Response ');
```

OUTPUT:

Filter coefficients for BSF=

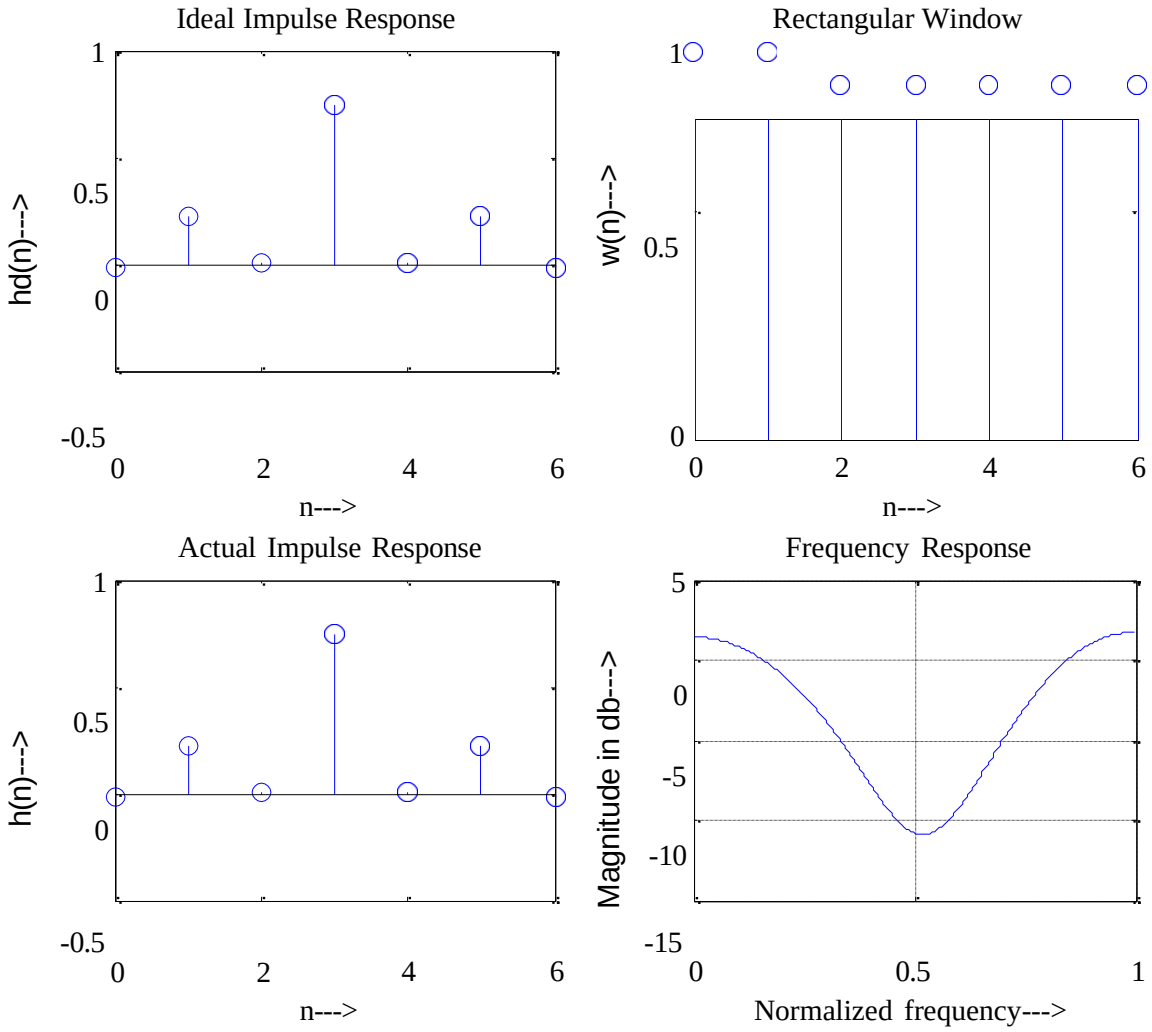
hd= -0.0173 0.2280 0.0072 0.7454 0.0072 0.2280 -0.0173

w = 1 1 1 1 1 1 1

h = -0.0173 0.2280 0.0072 0.7454 0.0072 0.2280 -0.0173

MODEL GRAPH:

BANDSTOP FILTER



RESULT:

Thus a program to design the FIR low pass, high pass, band pass and band stop filters was Written and response of the filter using MATLAB was executed.

EX. NO:6a	**DESIGN OF IIR FILTER by BILINEAR TRANSFORMATION METHOD**
DATE:	

AIM:

To design IIR Filter by Bilinear Transformation Methods using MATLAB 7.0.4

ALGORITHM:

1. Get the Passband and Stopband frequencies in rad/sec.
2. Get the Passband and Stopband ripples in dB.
3. Get the Sampling Frequency.
4. Compute the order of the filter.
5. Obtain the numerator and denominator coefficients of the filter function. 6. Plot the Frequency response of the filter using Bilinear Transformation. 7. Specify suitable X-axis and Y-axis label.
8. Specify suitable title for the signal.
9. If needed, Grid can be turned on.

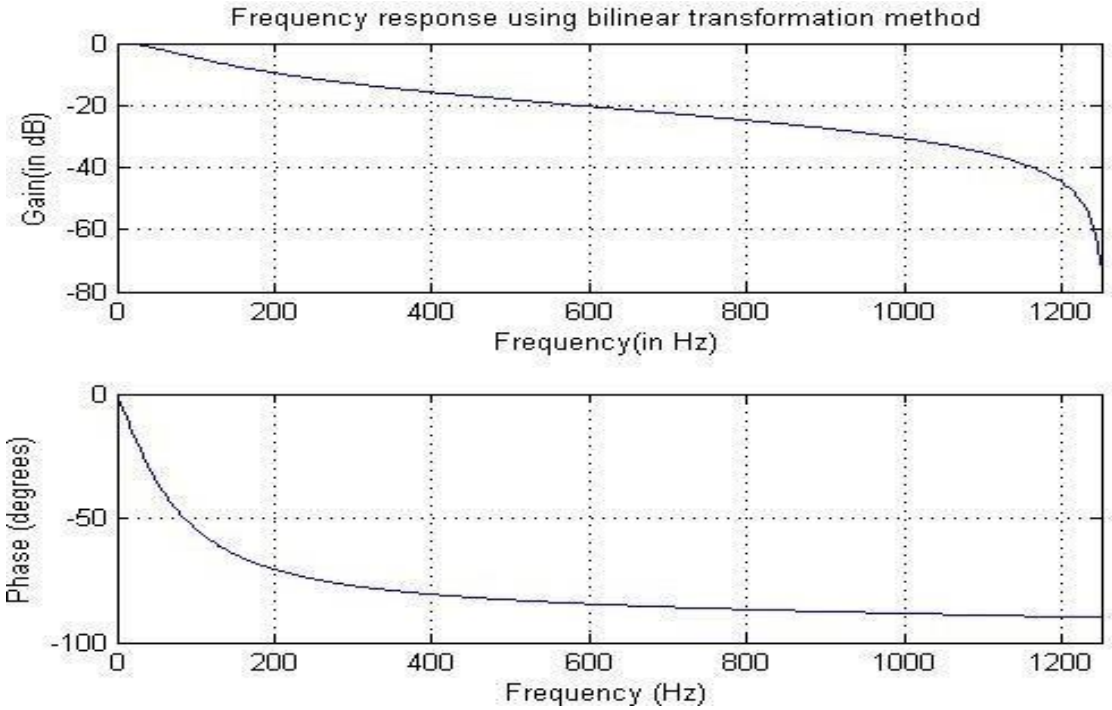
PROGRAM:

```
clc;
clear;
wp=input('Enter the passband edge frequency in rad/sec:');
ap=input('Enter the passband ripple in dB:');
ws=input('Enter the stopband edge frequency in rad/sec:');
as=input('Enter the stopband ripple in dB:');
fs=input('Enter the sampling frequency:');
[n,wn]=buttord(wp,ws,ap,as,'s');
disp('The order of the filter is N=');
[num,den]=butter(n,wn,'s');
[b,a]=bilinear(num,den,fs);
freqz(b,a,512,fs);
grid on;
xlabel('Frequency(in Hz)');
ylabel('Gain(in dB)');
title('Frequency response using bilinear transformation
method');
```

INPUT:

```
Enter the passband frequency in rad/sec:63
Enter the passband ripple in db:4
Enter the stopband frequency in rad /sec:2512.27
Enter the stopband ripple in db:15
Enter the sampling frequency:2500
```

OUTPUT:



RESULT :

Thus the IIR Filter was designed successfully by Bilinear Transformation Method using MATLAB 7.0.4

EX. NO:6b	DESIGN OF IIR FILTER by IMPULSE INVARIANT TRANSFORMATION METHOD
DATE:	

AIM :

To design IIR Filter by Impulse Invariant Transformation Methods using MATLAB 7.0.4

ALGORITHM :

1. Get the Passband and Stopband frequencies in rad/sec.
2. Get the Passband and Stopband ripples in dB.
3. Get the Sampling Frequency.
4. Compute the order of the filter.
5. Obtain the numerator and denominator coefficients of the filter function. 6. Plot the Frequency response of the filter using Impulse Invariant Transformation. 7. Specify suitable X-axis and Y-axis label.
8. Specify suitable title for the signal.
9. If needed, Grid can be turned on.

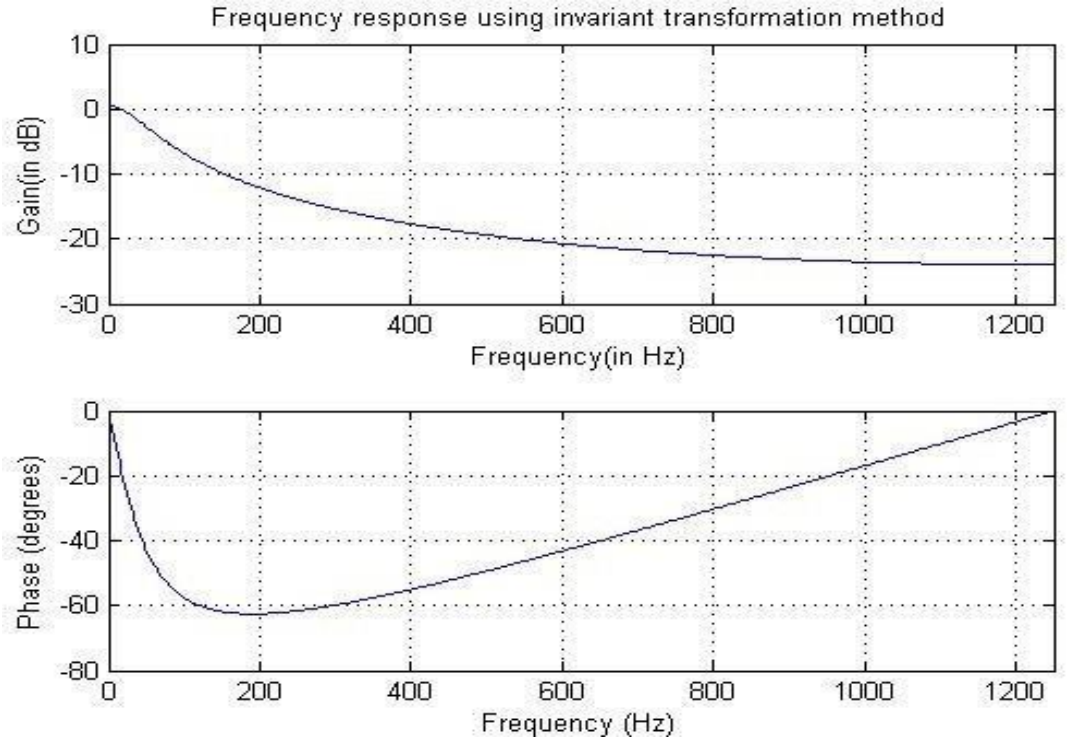
PROGRAM:

```
clc;
clear;
wp=input('Enter the passband edge frequency in rad/sec:');
ap=input('Enter the passband ripple in dB:');
ws=input('Enter the stopband edge frequency in rad/sec:');
as=input('Enter the stopband ripple in dB:');
fs=input('Enter the sampling frequency:');
[n,wn]=buttord(wp,ws,ap,as,'s');
disp('The order of the filter is N=');
[num,den]=butter(n,wn,'s');
[b,a]=impinvar(num,den,fs);
freqz(b,a,512,fs);
grid on;
xlabel('Frequency(in Hz)');
ylabel('Gain(in dB)');
title('Frequency response using invariant transformation method');
```

INPUT:

Enter the passband frequency in rad/sec:65
Enter the passband ripple in db:3
Enter the stopband frequency in rad/sec:2354.07
Enter the stopband ripple in db:18
Enter the sampling frequency:2500

OUTPUT:



RESULT :

Thus the IIR Filter was designed successfully by Impulse Invariant Transformation Method using MATLAB 7.0.4

EX. NO. :7	IMPLEMENTATION OF ERROR DETECTION / ERROR CORRECTION TECHNIQUES
DATE:	

AIM:

To write a C program for Implementation of Error Detection and Error correction Techniques.

SOFTWARE REQUIREMENTS:

Turbo C

ALGORITHM:

1. Start the program.
2. Get the input bits values from the user
3. To find out the error and hamming code based on the user request.
4. To send error in check bit to the client side.
5. Display the correct code.
6. Stop the program

PROGRAM:

```
#include<stdio.h>
#include<math.h>
Void main ()
{
        int i, a[4],c[3],r[7], clk[3], n, sum=0;
        printf ("Enter data bits\n");
        for (i=3;i>=0;i--)
        scanf("%d",&a[i]);
        printf("\n");
        c[0]=(a[0]+a[1]+a[2])%2;
        c[1]=(a[1]+a[2]+a[3])%2;
        c[2]=(a[1]+a[0]+a[3])%2;
        printf ("data bits after hamming code is\n");
        for(i=3;i>=0;i--)
        printf("%d",a[i]);
        for(i=2;i>=0;i--)
        printf("%d",c[i]);
        printf("Enter received code\n");
        for(i=0;i<7;i++) scanf("%d",&r[i]);
        clk[0]=(r[3]+r[1]+r[2]+r[6])%2;
        clk[1]=(r[0]+r[2]+r[1]+r[5])%2;
        clk[2]=(r[0]+r[2]+r[3]+r[4])%2;
        sum=4*clk[2]+2*clk[1]+1*clk[0];
        if(sum==0)
                printf("\n u have received correct code\n");
        if(sum==1)
        {
                printf("Error in check bit 2\n");
                printf("The correct code is");
                r[6]=(r[6]+1)%2;
                for(i=0;i<7;i++)
```

```
        printf("%d",r[i]);
}
if(sum==2)
{
        printf("Error in check bit 1\n");
        printf("The correct code is");
        r[5]=(r[5]+1)%2;
        for(i=0;i<7;i++) printf("%d",r[i]);
}
if(sum==3)
{
        printf("\nError in data bit 1");
        printf("The correct code is");
        r[1]=(r[1]+1)%2;
        for(i=0;i<7;i++) printf("%d",r[i]);
}
if(sum==4)
{
        printf("\n Error in chect bit 0");
        printf("The correct code is");
        r[4]=(r[4]+1)%2;
        for(i=0;i<7;i++) printf("%d",r[i]);
}
if(sum==5)
{
        printf("\n Error in data bits 3");
        printf("The correct code is");
        r[3]=(r[3]+1)%2;
        for(i=0;i<7;i++) printf("%d",r[i]);
}
if(sum==6)
{
        printf("Error in data bits 0");
        printf("The correct code");
        r[0]=(r[0]+1)%2;
        for(i=0;i<7;i++); printf("%d",r[i]);
}
if(sum==7)
{
        printf("Error in data bits 2");
        printf("The correct code is");
        r[2]=(r[2]+1)%2;
        for(i=0;i<7;i++) printf("%d",r[i]);
}
```

OUTPUT:

```
Enter data bits 3 3 5 6 5
Data bits after hamming code is 3 3 5 6 5 1 1 1
Enter received code 7 6 6 8 9 0 0 5 Error
in check bit 1
The correct code is 7 6 6 8 9 0 1 5
```

RESULT:

Thus the c program for error correction and detection was executed and the results are verified successfully.

EX. NO. :8	IMPLEMENTATION OF STOP AND WAIT PROTOCOL AND SLIDING WINDOW
DATE:	

AIM :

To Implement Stop and Wait Protocol and sliding window protocol.

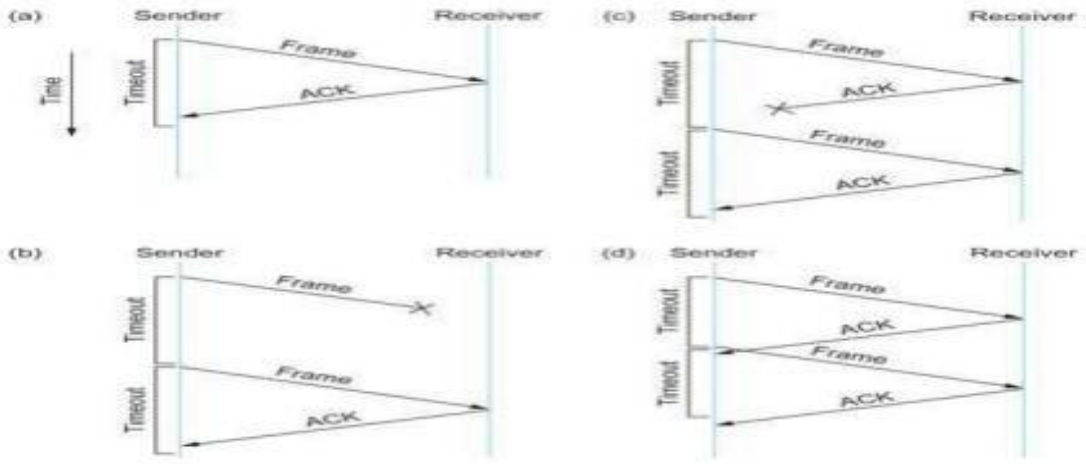
APPARATUS REQUIRED:

1. PENTIUM – PC
2. C COMPLIER
3. TURBO C

STOP AND WAIT PROTOCOL

THEORY:

The Stop-and-Wait protocol uses both flow and error control. In this protocol, the sender sends one frame at a time and waits for an acknowledgment before sending the next one. To detect corrupted frames, we need to add a CRC to each data frame. When a frame arrives at the receiver site, it is checked. If its CRC is incorrect, the frame is corrupted and silently discarded. The silence of the receiver is a signal for the sender that a frame was either corrupted or lost. Every time the sender sends a frame, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next frame (if it has one to send). If the timer expires, the sender resends the previous frame, assuming that the frame was either lost or corrupted. This means that the sender needs to keep a copy of the frame until its acknowledgment arrives. When the corresponding acknowledgment arrives, the sender discards the copy and sends the next frame if it is ready. Only one frame and one acknowledgment can be in the channels at any time. The advantages are Limited buffer size, sooner Errors detection and prevents one station occupying medium for long periods



ALGORITHM:

(i) Start Program

1. Start
2. Declare the required variables and file pointers.
3. Get the choice from the user if he needs a Selective Retransmission or Go-Back-N protocol.
4. Select "1" for Sending the data and "2" for the end of transmission
5. For sending the data open a text file in write mode and write the data that has to be sent.
6. Once written close the file.
7. Check the ack.txt file in which the acknowledgement from the receiver is stored.
8. If the acknowledgment is positive, then send the data to the receiver.
9. If all the data are sent, then select Option "2" to end the transmission.

(ii) Stop Program

1. Start
2. Declare the required variables and file pointers.
3. For receiving the data, open a data.txt text file in read mode and read the data that was sent by the receiver.
4. Open the ack.txt file in write mode and write as "Yes" if received correctly.
5. Else, write as 'No" in the ack.txt file.
6. Close the file.

Sender (sender.c):

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

#define PACKET_SIZE 1024

// Simulated packet loss rate (0% for a reliable channel)
#define PACKET_LOSS_RATE 0

// Function to simulate packet loss
bool isPacketLost() {
    return (rand() % 100) < PACKET_LOSS_RATE;
}

int main() {
    srand(time(NULL));

    // Simulated receiver's address
    int receiverAddress = 1;

    // Simulated channel
    int channelAddress = 2;

    // Data to be sent
```

```
    char data[] = "Hello, Receiver!";

    // Calculate the number of packets
    int dataLength = strlen(data);
    int numPackets = dataLength / PACKET_SIZE + 1;

    // Initialize sequence number
    int sequenceNumber = 0;

    // Simulated ACK
    int ackNumber = 0;

    for (int i = 0; i < numPackets; i++) {
        // Create a packet
        char packet[PACKET_SIZE];
        memset(packet, 0, PACKET_SIZE);
        int packetSize = (i == numPackets - 1) ? dataLength % PACKET_SIZE : PACKET_SIZE;
        memcpy(packet, data + i * PACKET_SIZE, packetSize);

        // Simulate packet loss
        if (isPacketLost()) {
            printf("Packet %d lost.\n", sequenceNumber);
            continue;
        }

        // Send the packet to the channel
        printf("Sending packet %d to channel.\n", sequenceNumber);

        // Simulate transmission delay
        usleep(1000);  // Sleep for 1 ms

        // Simulate ACK reception
        if (rand() % 4 == 0) {
            // Packet lost in the channel
            printf("Packet %d lost in the channel.\n", sequenceNumber);
        } else {
            // Packet received at the receiver
            printf("Packet %d received at the receiver.\n", sequenceNumber);

            // Simulate ACK transmission delay
            usleep(1000);  // Sleep for 1 ms

            // Send the ACK back to the sender
            printf("Sending ACK %d back to the sender.\n", ackNumber);
            ackNumber = 1 - ackNumber;  // Toggle the ACK number
        }

        // Move to the next sequence number
        sequenceNumber = 1 - sequenceNumber;
    }

    return 0;
}
```

Receiver (receiver.c):

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <time.h>

#define PACKET_SIZE 1024

// Simulated packet loss rate (0% for a reliable channel)
#define PACKET_LOSS_RATE 0

// Function to simulate packet loss
bool isPacketLost() {
    return (rand() % 100) < PACKET_LOSS_RATE;
}

int main() {
    srand(time(NULL));

    // Simulated sender's address
    int senderAddress = 0;

    // Simulated channel
    int channelAddress = 2;

    // Initialize sequence number
    int sequenceNumber = 0;

    // Simulated ACK
    int ackNumber = 0;

    while (1) {
        // Simulate packet loss
        if (isPacketLost()) {
            printf("Packet %d lost in the channel.\n", sequenceNumber);
        } else {
            // Packet received in the channel
            printf("Packet %d received in the channel.\n", sequenceNumber);

            // Simulate ACK transmission delay
            usleep(1000);  // Sleep for 1 ms

            // Send the ACK back to the sender
            printf("Sending ACK %d back to the sender.\n", ackNumber);
            ackNumber = 1 - ackNumber;  // Toggle the ACK number
        }

        // Move to the next sequence number
        sequenceNumber = 1 - sequenceNumber;
    }
```

```
    return 0;
}
```

Sender's Output:

Sending packet 0 to channel.
Packet 0 received at the receiver.
Sending ACK 0 back to the sender.
Sending packet 1 to channel.
Packet 1 lost.
Sending packet 0 to channel.
Packet 0 received at the receiver.
Sending ACK 1 back to the sender.

Receiver's Output:

Packet 0 received in the channel.
Sending ACK 0 back to the sender.
Packet 1 lost in the channel.
Packet 0 received in the channel.
Sending ACK 1 back to the sender.
Packet 1 received in the channel.
Sending ACK 0 back to the sender.

RESULT:

Thus the “Sliding window” and “Stop and Wait” protocol programmed using C is implemented successfully.

EX. NO. :9	IMPLEMENTATION OF IP ADDRESS CONFIGURATION.
DATE:	

AIM :

To implementation of IP address configuration.

APPARATUS REQUIRED:

1. Operating System : Windows NT/2000/XP or LINUX
2. Programming Tool :Network Simulator (NS2)

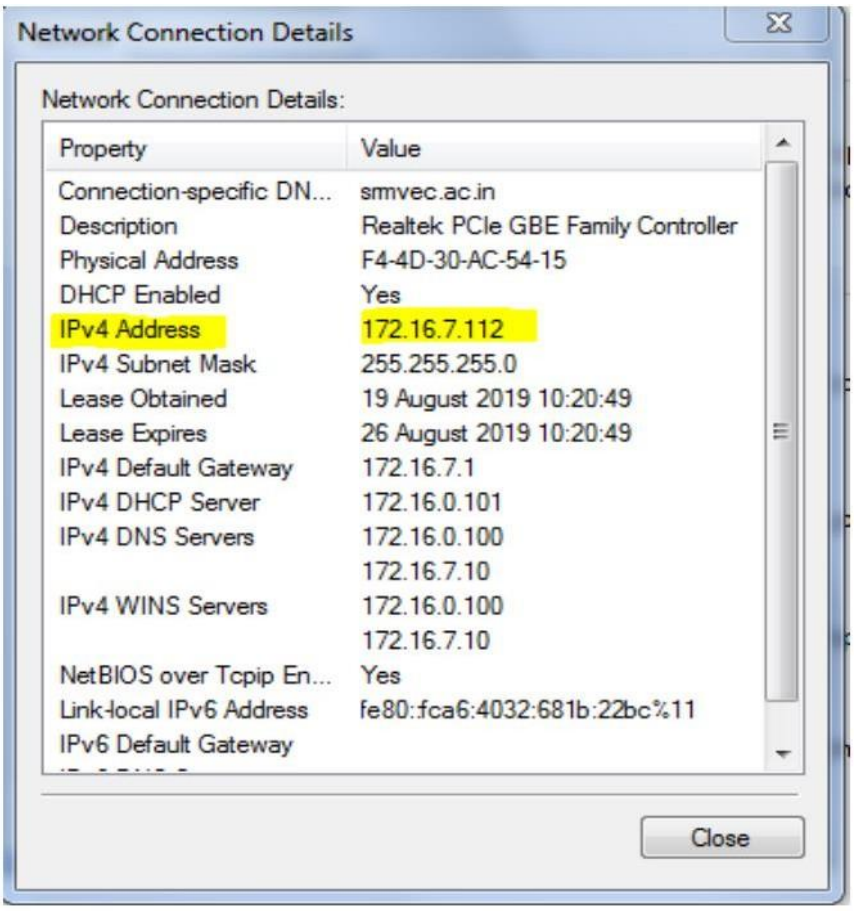
IP Address Configuration:

An Internet Protocol (IP) address is a unique number assigned to every device on a network. Just as a street address determines where a letter should be delivered, an IP address identifies computers on the Internet. Network devices use IP addresses to communicate with each other. IP addresses are required by any network adapter on any computer that needs to connect to the Internet or another computer. Addresses are given out to network computers in one of two manners, dynamically or statically.

To set a static IP address in Windows 7, 8, and 10:

1. Click Start Menu > Control Panel > Network and Sharing Center or Network and Internet >
2. Network and Sharing Center.
3. Click on Local Area Connection.
4. Click Details.
5. View for the Internet Protocol Version 4 (TCP/IPv4) address.

OUTPUT



RESULT:

Thus the IP address configuration was implemented successfully.

EX. NO. :10	DATA ENCRYPTION AND DECRYPTION USING RSA (RIVEST, SHAMIR AND ADLEMAN) ALGORITHM
DATE:	

AIM

To write a C program for implementation data encryption and decryption using

RSA (RIVEST, SHAMIR AND ADLEMAN) Algorithm.

SOFTWARE REQUIREMENTS:

Turbo C

ALGORITHM:

1. Start the program.
2. Get the input bits values from the user.
3. To generate private and public keys.
4. Display the private and public keys.
5. Execute the encryption and decryption process.
6. Stop the program.

PROGRAM :

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

// Function to calculate GCD (Greatest Common Divisor)
int gcd(int a, int b) {
while (b != 0) {
int temp = b;
b = a % b;
a = temp;
 }
return a;
 }

// Function to calculate modular exponentiation (M^E mod N)
long long modExpo(long long base, long long exp, long long mod) {
long long result = 1;
base = base % mod;
while (exp > 0) {
if (exp % 2 == 1)  // If exp is odd, multiply base with result
result = (result * base) % mod;
exp = exp >> 1;  // Divide exp by 2
base = (base * base) % mod;
}
return result;
```

```
 }

// Function to find modular inverse (d) using brute force method
int modInverse(int e, int phi) {
int d = 1;
while ((e * d) % phi != 1) {
d++;
}
return d;
}

void main() {
clrscr();

int p = 17, q = 19;  // Select two prime numbers
int n = p * q;
int phi = (p - 1) * (q - 1);
int e = 2;

// Finding suitable e (encryption key)
while (e < phi) {
if (gcd(e, phi) == 1)
break;
else
e++;
}

int d = modInverse(e, phi);  // Compute decryption key

printf("RSA Public Key: { %d , %d }\n", e, n);
printf("RSA Private Key: { %d , %d }\n", d, n);

// Encryption Process
int message;
printf("\nEnter message to encrypt (integer format): ");
scanf("%d", &message);

long long encryptedMessage = modExpo(message, e, n);
printf("Encrypted Message: %lld\n", encryptedMessage);

// Decryption Process
long long decryptedMessage = modExpo(encryptedMessage, d, n);
printf("Decrypted Message: %lld\n", decryptedMessage);

getch();
}
```

Explanation:

1. **Key Generation:**
    - The program selects two prime numbers p = 17 and q = 19.
    - It computes n = p * q and **Euler's totient function** φ(n) = (p-1) * (q-1).
    - The encryption exponent e is chosen such that **gcd(e, φ(n)) = 1**.
    - The decryption exponent d is found using modular inverse: e×dmod ϕ(n)=1e \times d \mod \phi(n) = 1e×dmodϕ(n)=1.
2. **Encryption:**
    - The user enters a **plaintext message** as an integer.
    - The **ciphertext** is computed using C = M^e mod n.
3. **Decryption:**
    - The encrypted message is decrypted using M = C^d mod n

OUTPUT:

```
RSA Public Key: { 5 , 323 }
RSA Private Key: { 77 , 323 }
Enter message to encrypt (integer format): 12
Encrypted Message: 248
Decrypted Message: 12
```

RESULT:

Thus the C program for data encryption and decryption was executed and the results are Verified successfully.

EX. NO. :11	NETWORK TOPOLOGY - STAR, BUS, RING.
DATE:	

AIM:

To create scenario and study the performance of Star, Bus, Ring protocol through Cisco Packet tracer.

SOFTWARE:

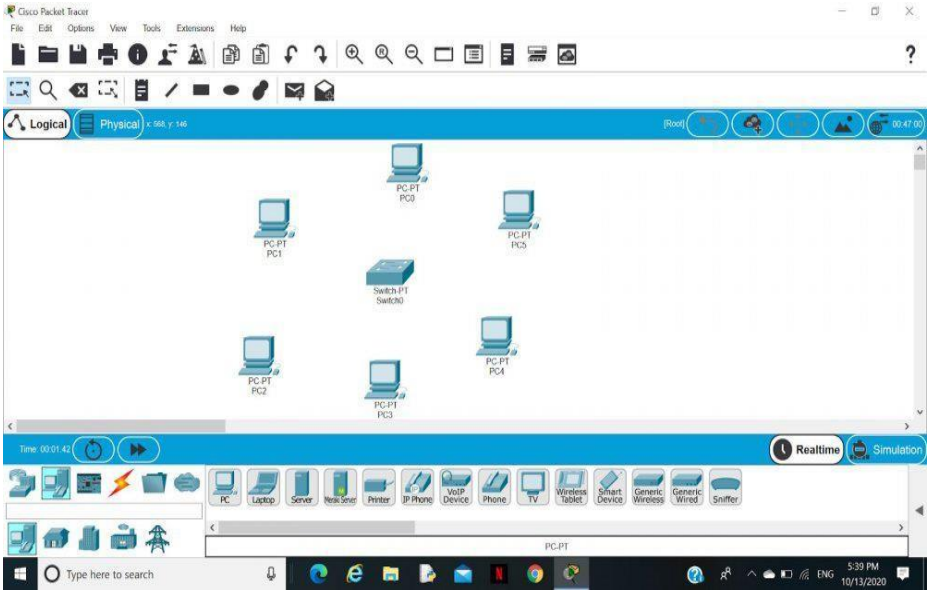
- Windows pc
- CISCO Packet Tracer Software

PROCEDURE:

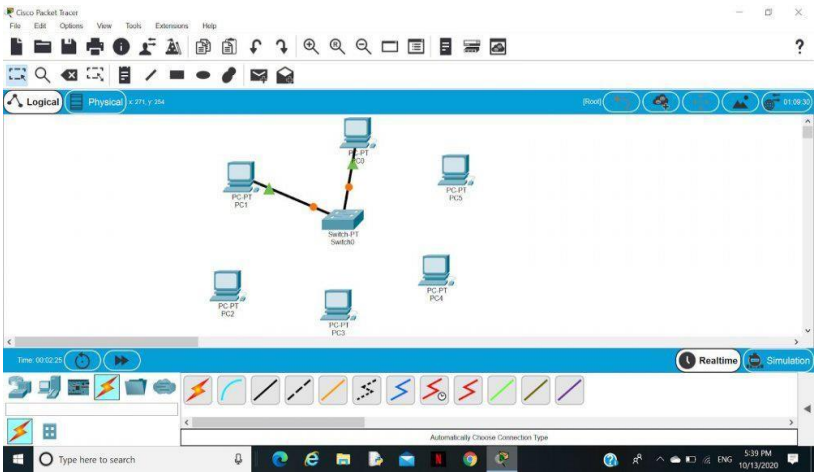
1. Open the CISCO Packet tracer software
2. Drag and drop 3 pcs using End Device Icons on the left corner
3. Select 8 port switch from switch icon list in the left bottom corner
4. Make the connections using Straight through Ethernet cables

STEPS IMPLEMENTING STAR TOPOLOGY USING CISCO PACKET TRACER:

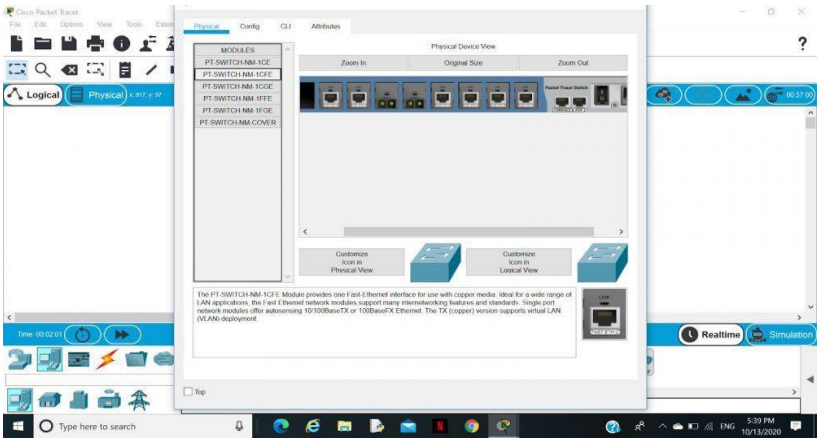
Step 1: We have taken a switch and linked it to six end devices.



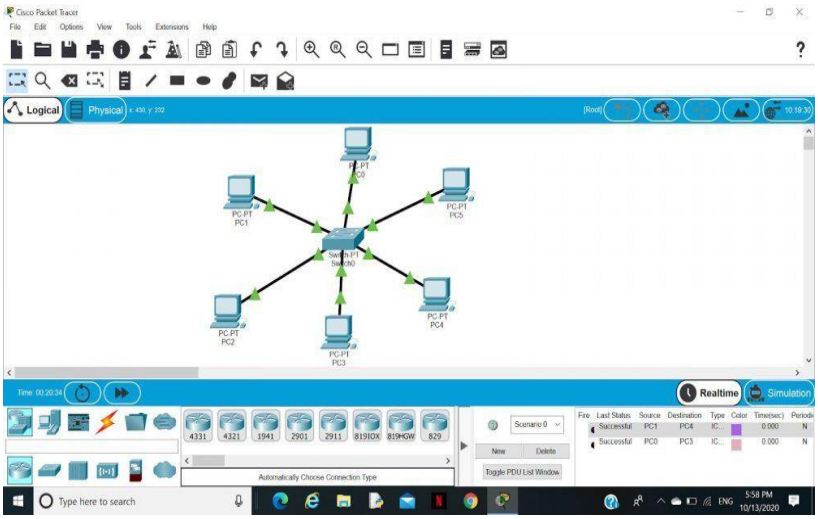
Step 2: Link every device with the switch



Step 3: Provide the IP address to each device



Step 4: Transfer message from one device to another and check the Table for Validation.



Steps to Configure and Setup Ring Topology in Cisco Packet Tracer :

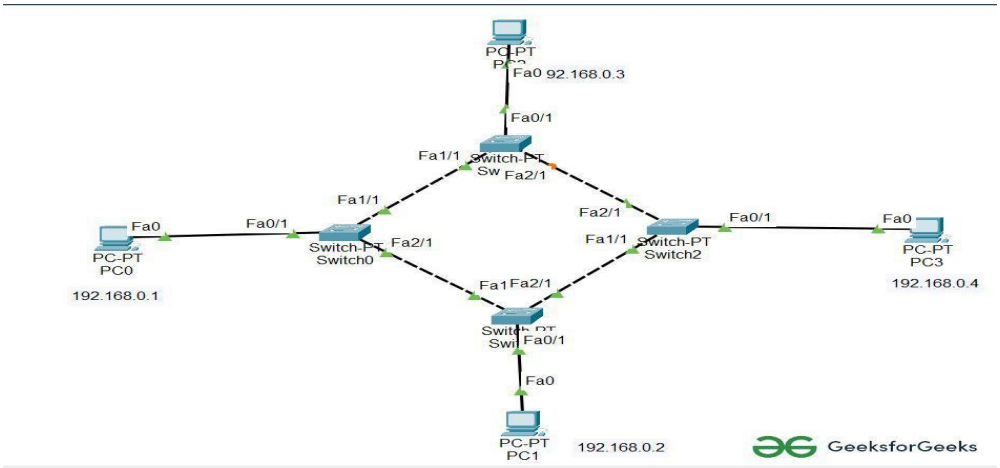
Step 1: First, open the cisco packet tracer desktop and select the devices given below:

S.NO	Device	Model Name
1.	PC	PC
2.	Switch	PT-Switch

IP Addressing Table

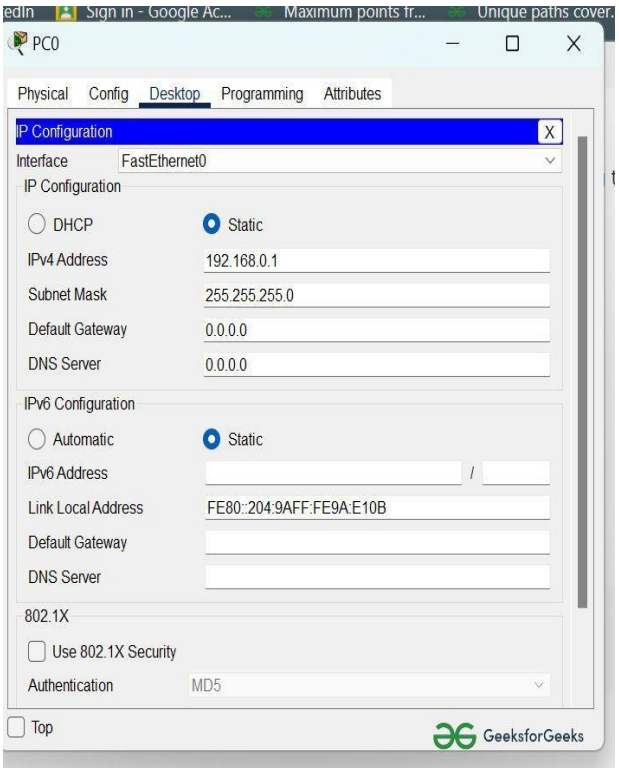
S.NO	Device	IPv4 Address	Subnet Mask
1.	**pc0**	192.168.0.1	255.255.255.0
2.	**pc1**	192.168.0.2	255.255.255.0
3.	**pc2**	192.168.0.3	255.255.255.0
4.	**pc3**	192.168.0.4	255.255.255.0

- Then, create a network topology as shown below the image.
- Use an Automatic connecting cable to connect the devices with others.

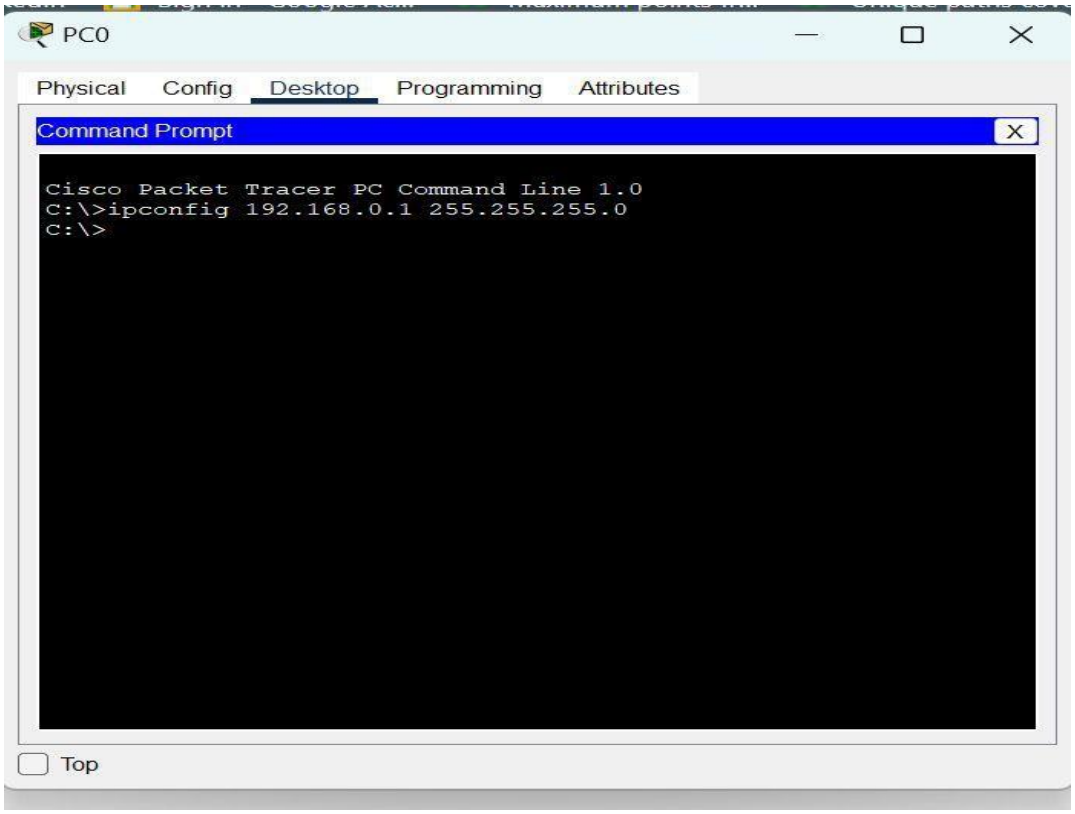


Step 2: Configure the PCs (hosts) with IPv4 address and Subnet Mask according to the IP addressing table given above.

- To assign an IP address in PC0, click on PC0.
- Then, go to desktop and then IP configuration and there you will IPv4 configuration.
- Fill IPv4 address and subnet mask.



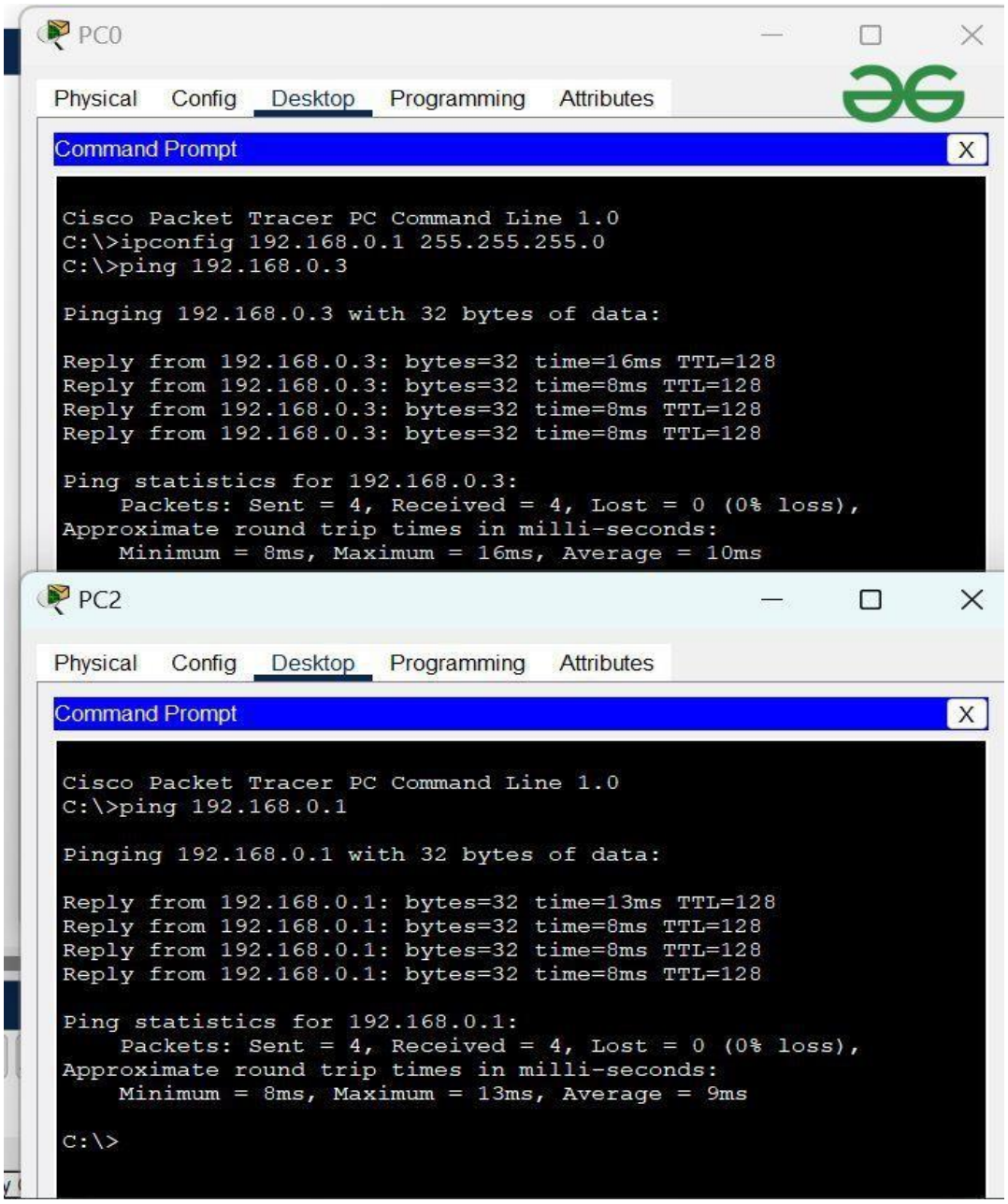
- Assigning IP address using the ipconfig command, or we can also assign an IP address with the help of a command.
- Go to the command terminal of the PC.
- Then, type ipconfig <IPv4 address><subnet mask><default gateway>(if needed) Example: ipconfig 192.168.0.1 255.255.255.0



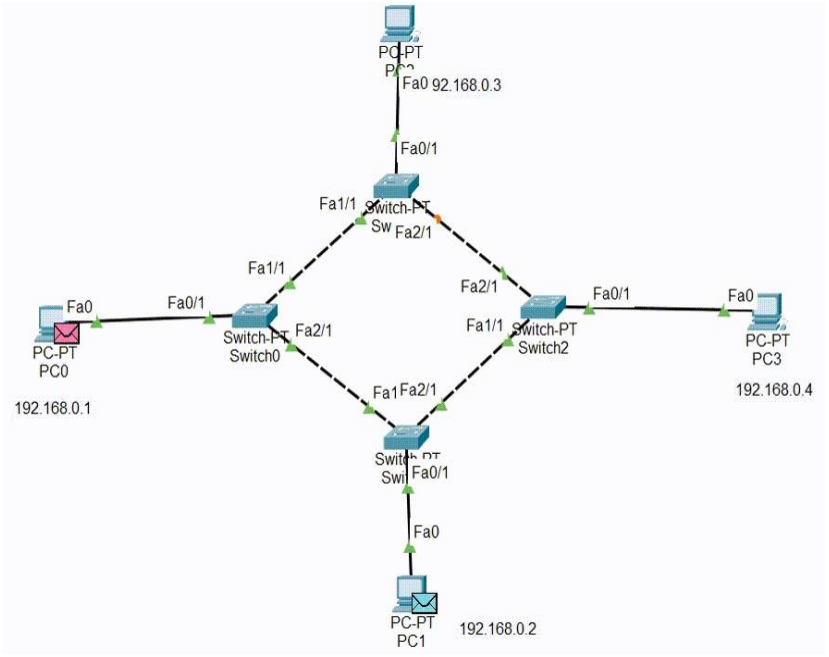
- Repeat the same procedure with other PCs to configure them thoroughly.

Step 3:

- Verify the connection by pinging the IP address of any host in PC0.
- Use the ping command to verify the connection.
- As we can see we are getting replies from a targeted node on both PCs.
- Hence the connection is verified.



- A simulation of the experiment is given below we have sent two PDU packets one targeted from PC0 to PC2 and another targeted from PC1 to PC3.



Steps to Configure and Setup Bus Topology in Cisco Packet Tracer :

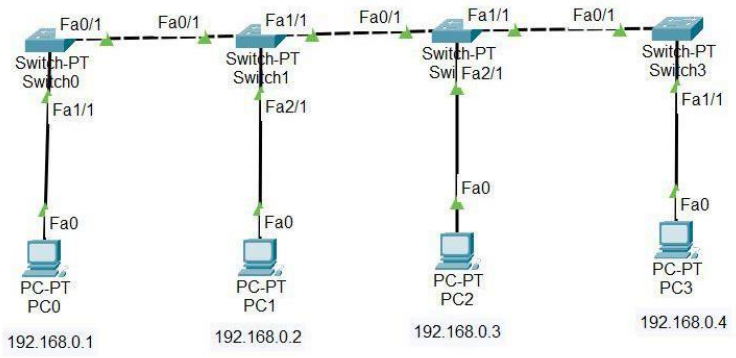
Step 1: First, open the cisco packet tracer desktop and select the devices given below:

S.NO	Device	Model-Name
1.	PC	PC
2.	Switch	PT-Switch

IP Addressing Table

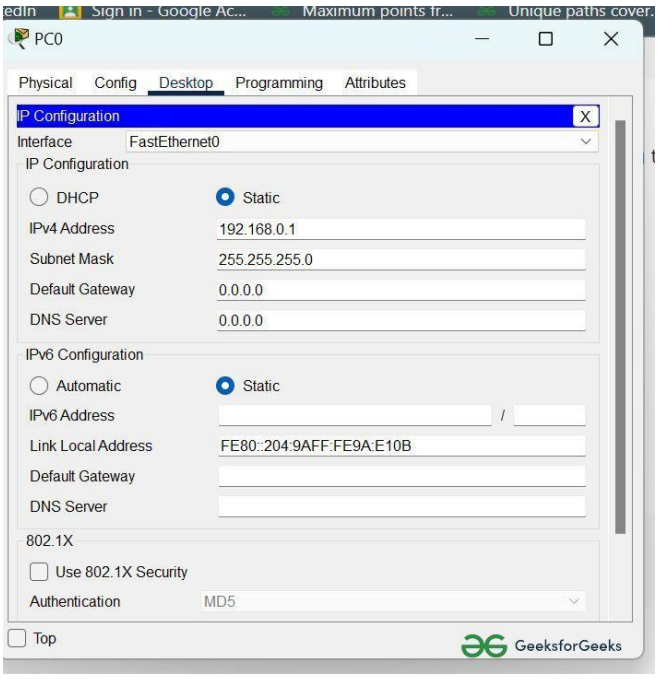
S.NO	Device	IPv4 Address	Subnet Mask
	pc0	192.168.0.1	255.255.255.0
	pc1	192.168.0.2	255.255.255.0
	pc2	192.168.0.3	255.255.255.0
	pc3	192.168.0.4	255.255.255.0

- Then, create a network topology as shown below image:
- Use an Automatic connecting cable to connect the devices with others.

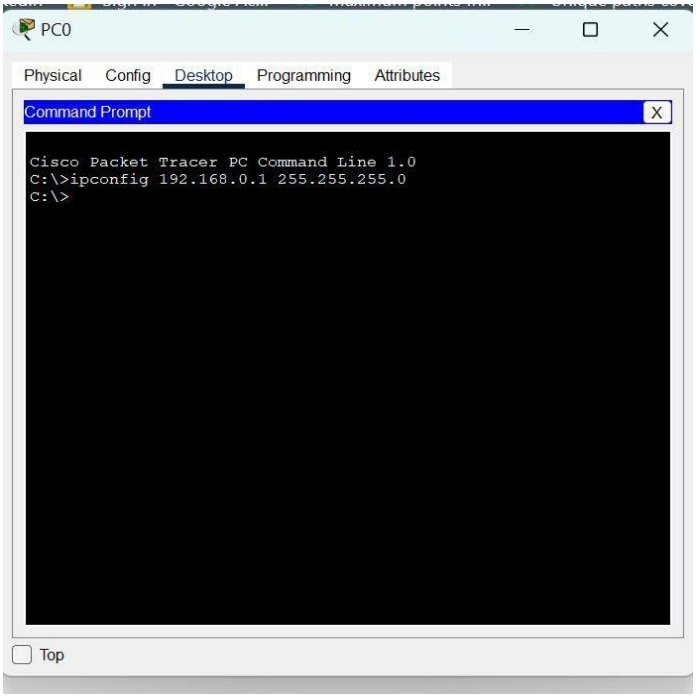


Step 2: Configure the PCs (hosts) with IPv4 address and Subnet Mask according to the IP addressing table given above.

- To assign an IP address in PC0, click on PC0.
- Then, go to desktop and then IP configuration and there you will IPv4 configuration.
- Fill IPv4 address and subnet mask.



- Assigning an IP address using the ipconfig command, or we can also assign an IP address with the help of a command.
- Go to the command terminal of the PC.
- Then, type ipconfig <IPv4 address><subnet mask><default gateway>(if needed) Example: ipconfig 192.168.0.1 255.255.255.0



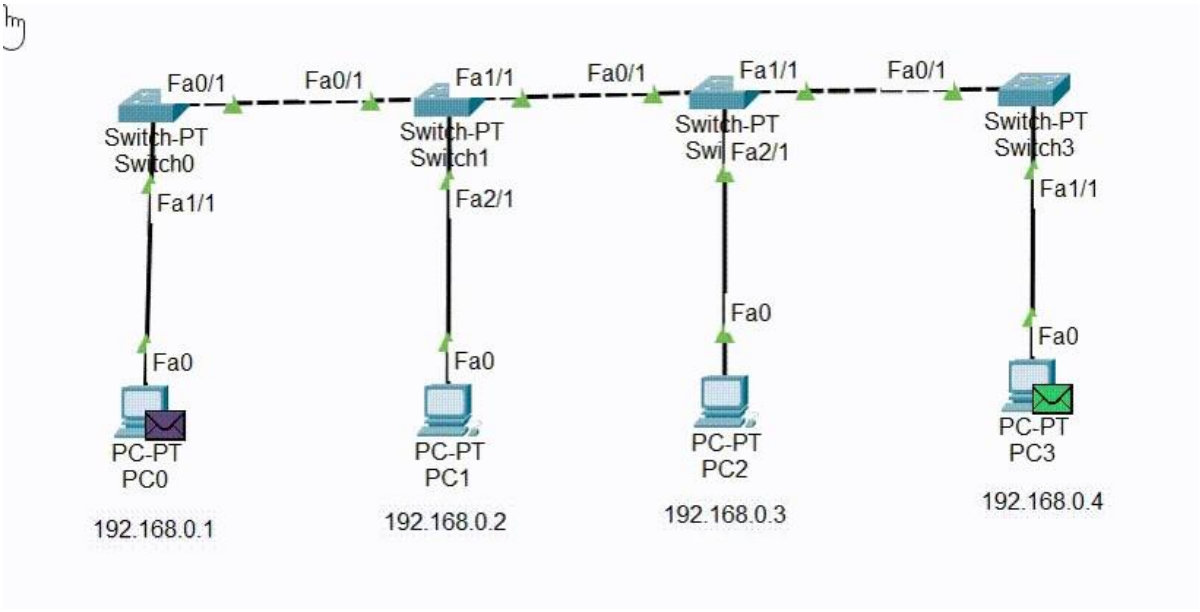
- Repeat the same procedure with other PCs to configure them thoroughly.

Step 3:

- Verify the connection by pinging the IP address of any host in PC0.
- Use the ping command to verify the connection.
- As we can see we are getting replies from a targeted node on both PCs.
- Hence the connection is verified.

Simulation Result:

A simulation of the experiment is given below we have sent two PDU packets one targeted from PC0 to PC2 and another targeted from PC3 to PC1.



RESULT:

Thus the Network Topology of Star ,Ring ,Bus was implemented and the output was verified using Cisco Packet Tracer.

EX. NO. :12	IMPLEMENTATION OF LINK STATE ROUTING ALGORITHM
DATE:	

AIM:

To implement link state routing algorithm.

APPARATUS REQUIRED:

1. PENTIUM – PC
2. C COMPLIER
3. TURBO C

PRINCIPLE:

Link state routing works on the following principle.

1. Discover the neighbour and keep their network address.
2. Measure the delay or cost to each of its neighbour.
3. Construct a packet telling all it has just learned.
4. Send the packet to all router.
5. Compute the shortest path to every router.

LINK STATE ROUTING:

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

#define MAX_NODES 10

// Structure to represent a link between two nodes
typedef struct {
    int node1;
    int node2;
    int cost;
} Link;

// Structure to represent a node's state
typedef struct {
    int id;
    int distance[MAX_NODES];
} NodeState;

// Function to initialize the node state
```

```
void initializeNodeState(NodeState *node, int nodeCount) {
    node->id = nodeCount;
    for (int i = 0; i < MAX_NODES; i++) {
        node->distance[i] = (i == nodeCount) ? 0 : -1;
    }
}

// Function to update the routing table using link-state information
void updateRoutingTable(NodeState nodes[], Link links[], int nodeCount, int linkCount) {
    for (int i = 0; i < nodeCount; i++) {
        for (int j = 0; j < linkCount; j++) {
            int node1 = links[j].node1;
            int node2 = links[j].node2;
            int cost = links[j].cost;

            if (nodes[i].distance[node1] == -1 || nodes[i].distance[node1] + cost < nodes[i].distance[node2]) {
                nodes[i].distance[node2] = nodes[i].distance[node1] + cost;
            }
        }
    }
}

// Function to print the routing table for all nodes
void printRoutingTable(NodeState nodes[], int nodeCount) {
    printf("Routing Table:\n");
    for (int i = 0; i < nodeCount; i++) {
        printf("Node %d: ", i);
        for (int j = 0; j < nodeCount; j++) {
            printf("To %d: %d\t", j, nodes[i].distance[j]);
        }
        printf("\n");
    }
}

int main() {
    int nodeCount = 5;
    int linkCount = 7;

    Link links[] = {
        {0, 1, 1},
        {0, 2, 3},
        {1, 2, 1},
        {1, 3, 4},
        {2, 3, 1},
        {2, 4, 2},
        {3, 4, 3}
    };

    NodeState nodes[nodeCount];

    for (int i = 0; i < nodeCount; i++) {
        initializeNodeState(&nodes[i], i);
    }
```

```
    updateRoutingTable(nodes, links, nodeCount, linkCount);
    printRoutingTable(nodes, nodeCount);

    return 0;
}
```

Output:

```
Routing Table:
Node 0: To 0: 0   To 1: 1   To 2: 2   To 3: 3   To 4: 5
Node 1: To 0: 1   To 1: 0   To 2: 1   To 3: 4   To 4: 5
Node 2: To 0: 2   To 1: 1   To 2: 0   To 3: 1   To 4: 3
Node 3: To 0: 3   To 1: 4   To 2: 1   To 3: 0   To 4: 2
Node 4: To 0: 5   To 1: 5   To 2: 3   To 3: 2   To 4: 0
```

RESULT:

Thus the link state algorithm was implemented and the output was verified.