Department of Information Technology
BABU BANARASI DAS
Institute of Technology & Management, Lucknow
Recognized by AICTE, Govt.of India & Affiliated to Dr. A.P.J Abdul Kalam Technical University,Lucknow,AKTU College Code=054

LAB MANUAL
OF
COMPUTER NETWORKS

[BCS – 653]

B.TECH, 3RD Year, Semester -VII

Dr. A.P.J. Abdul Kalam Tech. University
Uttar Pradesh

2025-26

Department of Information Technology

Faculty Name: Mrs.Soumya Saxena

(Assistant Professor)

Approved by: (Head of the Department)

MANUAL CONTENTS

This manual is intended for the 3rd year students of Information Technology in the subject of Computer Network Lab. This manual typically contains practical/lab sessions related Computer Network covering various aspects related to the subject of enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions.

PREFACE

This practical manual will be helpful for students of Information Technology for understanding the course from the point of view of applied aspects. Though all the efforts have been made to make this manual error free, yet some errors might have crept in inadvertently. Suggestions from the readers for the improvement of the manual are most welcomed.

VISION OF THE INSTITUTE

"To become a leading institute of providing professionally competent and socially responsive technocrats with high moral values."

MISSION OF THE INSTITUTE

- To create an eco-system for the dissemination of technical knowledge, to achieve academic excellence.
- To develop technocrats with creative skills and leadership qualities, to solve local and global challenges.
- To impart human values and ethics in students, to make them socially and Eco-friendly responsible

.

VISION OF THE DEPARTMENT

"To transform the students to be skilled IT professionals, Innovative leaders and environmentally receptive citizens"

MISSION OF THE DEPARTMENT

M1: To develop and deliver quality academic programs in emerging & innovative fields of engineering to empower the students to meet industry standards.

M2: To build students community with ethical standards to undertake R&D in thrust areas of national & international needs.

M3: To inculcate professional behavior, positive attitude and communication skills.

.

Program Educational Objectives(PEOs) of Department

PEO1: To develop the ability among students to understand the concept of core Information Technology subjects that will facilitate understanding of any new technology.

PEO2: To embed a strong foundation in the engineering fundamentals to solve analyze and design real time engineering products.

PEO3: To give exposures to emerging edge technologies adequate training and opportunities to work as a team on multi disciplinary projects with effective communication skills and leadership qualities.

Program Outcomes:(PO)

Graduates will be able to achieve

PO 1.**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 2.**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO 3.**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO 4.Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO 5.Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO 6.The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO 7.Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO 8.Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9.Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 10.Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear
PO 11.Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO 12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

Program Specific Outcomes(PSOs) of the Department

PSO1: Use and apply current technical concepts and practices in the core Information Technologies of Human Machine Learning, Information Management, Programming, Cloud Computing.

PSO2: Design, develop and test software systems for worldwide network of computers to provide solutions to real world problems.

Course Evaluation Scheme

Sr No	Subject Code	Subject Name	Periods			Evaluation Scheme				Total	Credit
			L	T	P	Sessional Assessment			PE		
						CT	TA	PS			
1.	BCS653	Computer Networks Lab	0	0	2	-	-	50	50	100	1

Course Objectives:

The teacher will explain:

1.	Able to learn brief knowledge about flow control protocols.
2.	Able to handle various error handling techniques.
3.	Able to analyze network routing and addressing techniques.

Pre- requisite:Fundamentals of Computer Science and Programming,Basic Algorithms.

Course Outcomes (COs)

Course Outcomes: The students should be able to:		**Bloom's Level**
CO1	Simulate different network topologies.	K3
CO2	Implement various framing methods of Data Link Layer.	K3
CO3	Implement various Error and flow control techniques.	K3
CO4	Implement network routing and addressing techniques.	K3
CO5	Implement transport and security mechanisms.	K3

CO-PO-PSO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	2	2	3	2	1	-	-	-	2	1	1	1	2	1
CO2	3	1	1	1	1	-	-	-	2	3	3	3	3	1
CO3	3	2	3	2	1	-	-	-	2	1	1	2	2	3
CO4	1	2	3	2	-	-	-	-	3	2	3	3	1	2
CO5	1	1	3	3	2	-	-	-	1	3	3	1	2	2
Avg.	2	2	3	2	1	0	0	0	2	2	2	2	2	2

The extent of mapping is as follows: 1 for low, 2 for moderate, 3 for high &"-" for No correlation between CO & PO.

COMPUTER NETWORK LAB (BCS – 653)
List of Experiments

Content Beyond Syllabus

S. No.	Name of the Experiment	Corresponding Cos
1.	To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc.	CO1
2.	Configuration of router, hub, switch etc.	CO 1
3.	Running and using services/commands like ping, traceroute, nslookup, arp, telnet, ftp, etc.	CO 3
4.	Write a program for bit Stuffing.	CO 2
5.	To Configure a Network Using Different Topologies	CO 2
6.	Write a program to compute Cyclic Redundancy Check.	CO 3
7.	To Implement the TCP/IP Model.	CO 4
8.	Implementation of Stop and Wait Protocol.	CO 4
9.	To write a program to find the Physical Address for a given IP addresses using Simulation.	CO 5
10.	Write a program to perform client sever chat application.	CO 5

11.	To connect two personal computer with straight thru and cross over twisted pair.	CO5
12.	Create a virtual network in GNS3 and configure VLANs, trunks, and inter-VLAN routing. Test the end-to-end connectivity for devices in different VLANs.	CO5

Soumya Saxena

Faculty Incharge

Experiment-1

Objective:
Study of different types of Network cables and Practically implement the cross-wiredcable and straight through cable using clamping tool.

Apparatus (Components): RJ-45 connector, Crimping Tool, Twisted pair Cable

Procedure:
To do these practical following steps should be done:

1. Start by stripping off about 2 inches of the plastic jacket off the end of the cable. Be verycareful at this point, as to not nick or cut into the wires, which are inside. Doing so could alterthe characteristics of your cable, or even worse render is useless. Check the wires, one more timefor nicks or cuts. If there are any, just whack the whole end off, and start over.

2. Spread the wires apart, but be sure to hold onto the base of the jacket with your other hand.
   You do not want the wires to become untwisted down inside the jacket. Category 5 cable mustonly have 1/2 of an inch of 'untwisted' wire at the end; otherwise it will be 'out of spec'. At thispoint, you obviously have ALOT more than 1/2 of an inch of un-twisted wire.

3. You have 2 end jacks, which must be installed on your cable. If you are using a pre-made
   able, with one of the ends whacked off, you only have one end to install - the crossed over end.

Below are two diagrams, which show how you need to arrange the cables for each type of cable
end. Decide at this point which end you are making and examine the associated picture below.

Experiment-2

Objective:
Study of following Network Devices in Detail
• Repeater
• Hub
• Switch
• Bridge
• Router
• Gate Way

Apparatus (Software): No software or hardware needed.

Procedure: Following should be done to understand this practical.

1. Repeater: Functioning at Physical Layer.Arepeater is an electronic device that receives

Diagram shows you how to prepare straight through wired connection

RJ45 Pin # (END 1)	Wire Color	Diagram End #1	RJ45 Pin # (END 2)	Wire Color	Diagram End #2
1	White/Orange		1	White/Green	
2	Orange		2	Green	
3	White/Green		3	White/Orange	
4	Blue		4	White/Brown	
5	White/Blue		5	Brown	
6	Green		6	Orange	
7	White/Brown		7	Blue	
8	Brown		8	White/Blue	

asignal and retransmits it at a higher level and/or higher power, or onto the other side of anobstruction, so that the signal can cover longer distances. Repeater have two ports ,so cannot beuse to connect for more than two devices.

2. Hub: An Ethernet hub, active hub, network hub, repeater hub, hub or concentratoris a device for connecting multiple twisted pair or fiber optic Ethernet devices together andmaking them act as a single network segment. Hubs work at the physical layer (layer 1) of theOSI model. The device is a form of multiport repeater. Repeater hubs also participate in collisiondetection, forwarding a jam signal to all ports if it detects a collision.

3. Switch: Anetwork switch or switching hub is a computer networking device that connectsnetwork segments.The term commonly refers to a network bridge that processes and routes dataat the data link layer (layer 2) of the OSI model. Switches that additionally process data at thenetwork layer (layer 3 and above) are often referred to as Layer 3 switches or multilayerswitches.

4. Bridge: A network bridge connects multiple network segments at the data link layer (Layer2) of the OSI model. In Ethernet networks, the term bridge formally means a device that behavesaccording to the IEEE 802.1D standard. A bridge and switch are very much alike; a switch beinga bridge with numerous ports. Switch or Layer 2 switch is often used interchangeably withbridge .Bridges can analyze incoming data packets to determine if the bridge is able to send thegiven packet to another segment of the network.

5. Router: A router is an electronic device that interconnects two or more computer networks,and selectively interchanges packets of data between them. Each data packet contains addressinformation that a router can use to determine if the source and destination are on the samenetwork, or if the data packet must be transferred from one network to another. Where multiplerouters are used in a large collection of interconnected networks, the routers exchangeinformation about target system addresses, so that each router can build up a table showing thepreferred paths between any two systems on the interconnected networks.

6. Gate Way: In a communications network, a network node equipped for interfacing withanother network that uses different protocols.

• A gateway may contain devices such as protocol translators, impedance matchingdevices, rate converters, fault isolators, or signal translators as necessary to providesystem interoperability. It also requires the establishment of mutually acceptableadministrative procedures between both networks.

• A protocol translation/mapping gateway interconnects networks with different networkprotocol technologies by performing the required protocol conversions.

Experiment-3

Objective:

Running and using services/commands like ping, traceroute, nslookup, arp, telnet, ftp, etc. (Study of basic network command and Network configuration commands.)

Apparatus (Software): Command Prompt And Packet Tracer.

Procedure: To do this EXPERIMENT- follows these steps:

In this EXPERIMENT- students have to understand basic networking commands e.g ping, tracert etc.

All commands related to Network configuration which includes how to switch to privilege mode

and normal mode and how to configure router interface and how to save this configuration to flash memory or permanent memory.

This commands includes

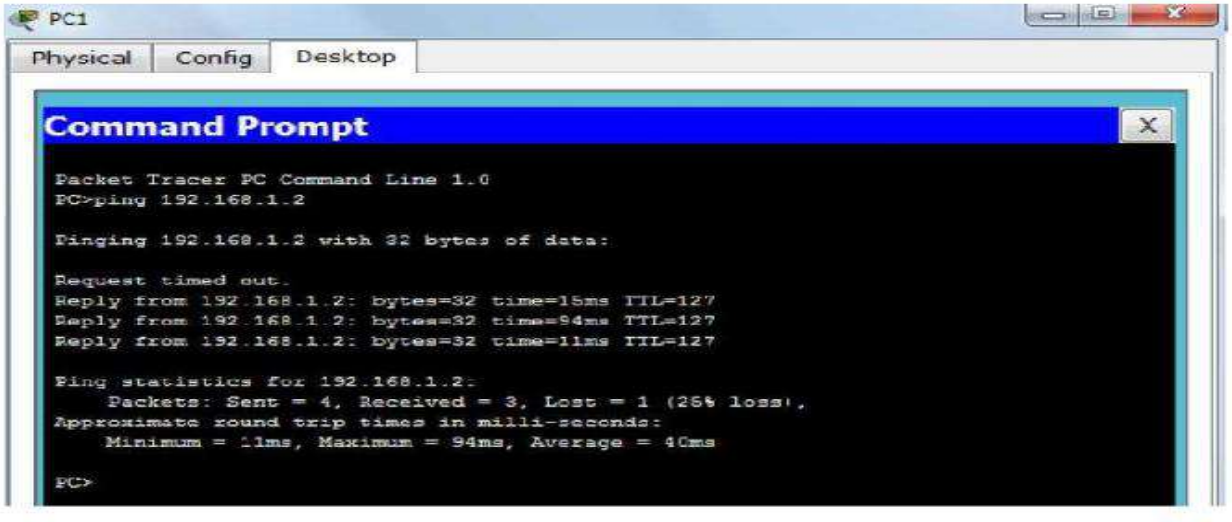
- Configuring the Router commands
- General Commands to configure network
- Privileged Mode commands of a router
- Router Processes & Statistics
- IP Commands
- Other IP Commands e.g. show ip route etc.

ping:

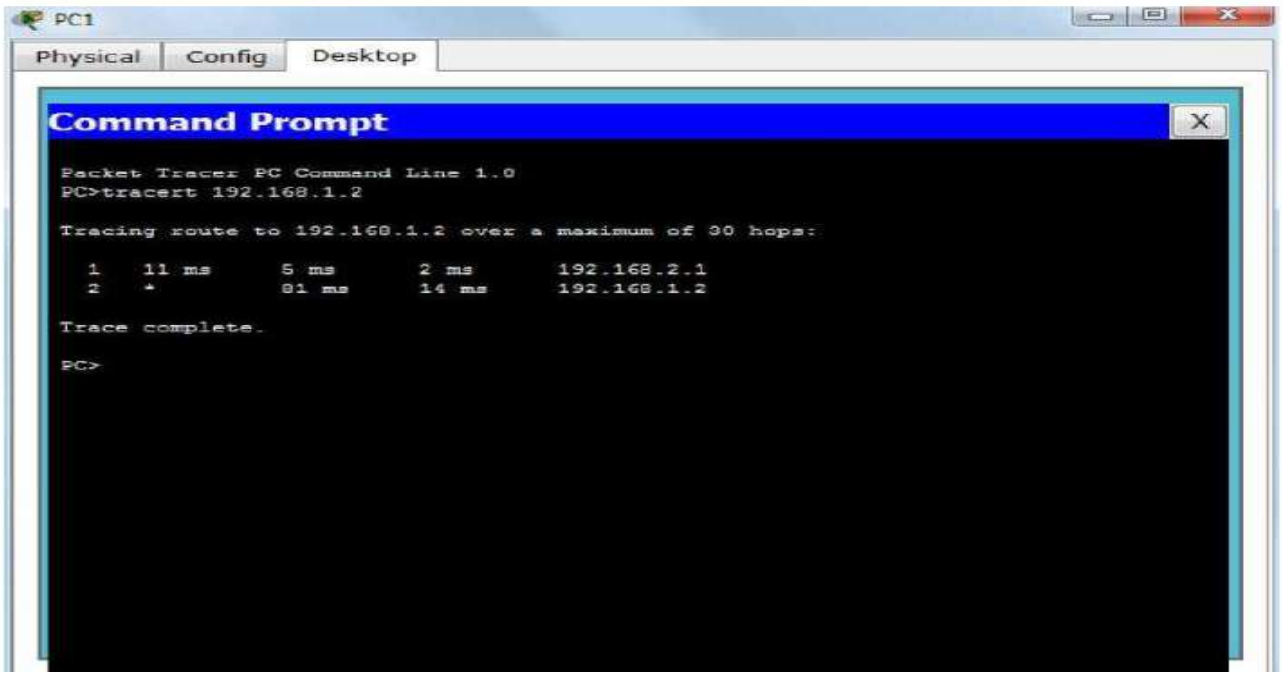
ping(8) sends an ICMP ECHO_REQUEST packet to the specified host. If the host responds, you

get an ICMP packet back. Sound strange? Well, you can “ping” an IP address to see if a machine

is alive. If there is no response, you know something is wrong.

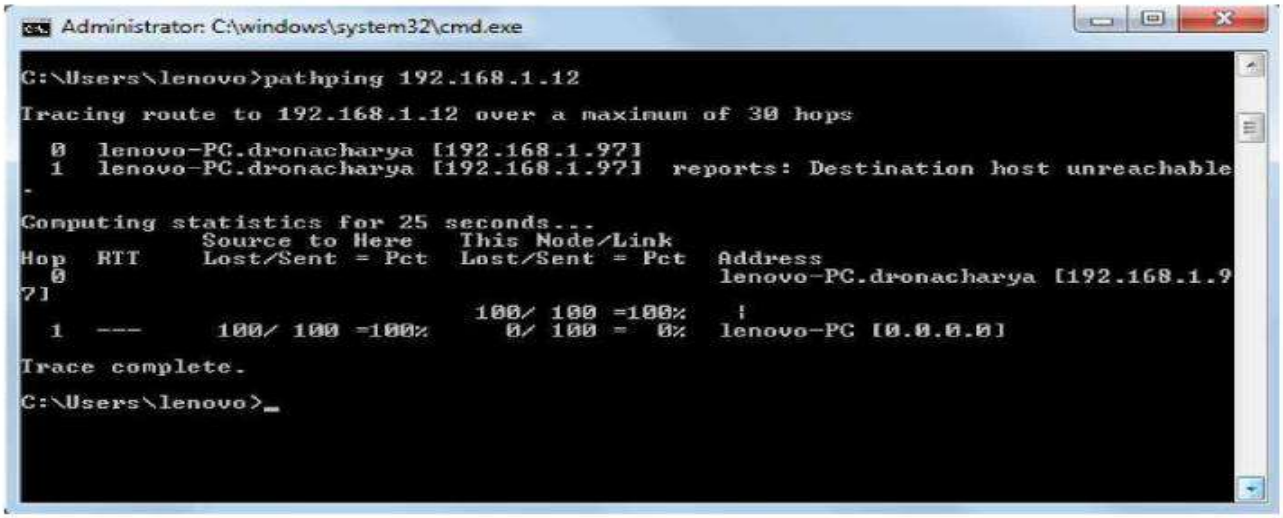


Traceroute:
Tracert is a command which can show you the path a packet of information takes from your computer to one you specify. It will list all the routers it passes through until it reaches its destination, or fails to and is discarded. In addition to this, it will tell you how long each 'hop' from router to router takes.



nslookup:
Displays information from Domain Name System (DNS) name servers.
NOTE :If you write the command as above it shows as default your pc's server name firstly.

pathping:
A better version of tracert that gives you statics about packet lost and latency.



Getting Help

In any command mode, you can get a list of available commands by entering a question mark (?).

Router>?

To obtain a list of commands that begin with a particular character sequence, type in those haracters followed immediately by the question mark (?).

Router#co?

configure connect copy

To list keywords or arguments, enter a question mark in place of a keyword or argument. Include a space before the question mark.

Router#configure ?

memory Configure from NV memory network Configure from a TFTP network host terminal Configure from the terminal

You can also abbreviate commands and keywords by entering just enough characters to make the command unique from other commands. For example, you can abbreviate the show command to sh.

Configuration Files

Any time you make changes to the router configuration, you must save the changes to memory

because if you do not they will be lost if there is a system reload or power outage. There are two

types of configuration files: the running (current operating) configuration and the startup configuration.

Use the following privileged mode commands to work with configuration files.

EXPERIMENT NUMBER: 4

Objective: Write a program to implement bit stuffing for a given binary file. That is stuff an extra‘0’ bit after continuous five 1’s in the file.

ALGORITHM:

Step 1: Start the program.
Step 2: Open the given input binary file in read mode and open the output file in write mode.
Step 3: Start reading the input binary file character by character and write it in the output binary file.
Step 4: Use previous, next and count to store the values of previous character, next character and number of ones respectively.
Step 5: If the count value is 5 then write one ‘0’ in the file in the current position of the output file and reset the value of count to zero.
Step 6: Continue steps 3 to 5 till the end of the file.
Step 7: Close the input and output files.
Step 8: Open the output binary file in read mode and display the characters from it.

Step 9: An extra '0' bit is inserted after continuous five 1's in the file.
Step 10: Close all the files.
Step 11: Stop the program.

CODING:

```
#include<stdio.h>
#include<conio.h>
void main()
{
FILE *fpt1, *fpt2;
char c, prev,next;
inti = 1;
clrscr();
fpt1 = fopen("in.txt","r");
fseek(fpt1, 0, SEEK_SET);
fpt2 = fopen("out.txt","w+");
if ((c = fgetc(fpt1))!=EOF)
{
fputc(c,fpt2);
prev = c;
}
do
{
next = fgetc(fpt1);
if (prev == '1'&& next == '1')
{
fputc(next,fpt2);
i++;
}
else
{
fputc(next,fpt2);
i = 1;
} if (
i==5)
fputc('0',fpt2);
else
{ prev = next; }
}while(next!=EOF);
fclose(fpt1);
fclose(fpt2);
fpt1 = fopen("in.txt","r");
printf("\n\nThe input file is: \n\n ");
do
{c= fgetc(fpt1);
putch(c);
}while(c!= EOF);
fclose(fpt1);
printf("\n\nThe output file is: \n\n ");
```

```
fpt2 = fopen("out.txt","r");
do
{ c= fgetc(fpt2);
putch(c);
}while(c!=EOF);
fclose(fpt2);
getch();
return 0;
}
```

OUTPUT:

The input file is:
10011111011111101010101111111100110
The output file is:
100111110011111010101010111110111100110

.

EXPERIMENT NUMBER: 5

Objective: To Configure a Network Using Different Topologies

Objective:

- To understand different network topologies.
- To configure networks using Cisco Packet Tracer.
- To verify communication between devices.

Requirements:

- Cisco Packet Tracer
- PC
- Switch
- Hub
- Router
- Ethernet Cable

Theory:

A network topology defines the physical or logical arrangement of devices in a computer network. The common topologies are Bus, Star, Ring, Mesh, and Tree.

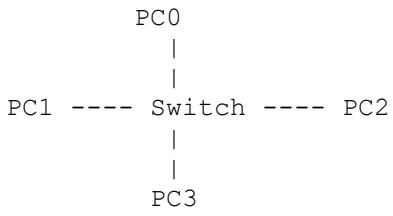
Procedure:

1. Open Cisco Packet Tracer.
2. Drag PCs and switches into the workspace.
3. Connect devices using suitable cables.
4. Assign IP addresses.
5. Test connectivity using the Ping command.

Observation:

- All devices were connected successfully.
- Ping reply was received.

Network Diagram



Step 1: Add Devices

- 1 Switch (2960)
- 4 PCs (PC0, PC1, PC2, PC3)

Step 2: Connect Devices

Use **Copper Straight-Through Cable**

Step 3: Configure IP Address

Click **PC → Desktop → IP Configuration**

Step 4: Test Connectivity

PC0 → Desktop → Command Prompt

ping 192.168.1.2

Result:
The network was successfully configured using different topologies.

```
Pinging 192.168.1.2 with 32 bytes of data:

Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128
Reply from 192.168.1.2: bytes=32 time<1ms TTL=128

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss)
```

EXPERIMENT NUMBER: 6

Objective: To find the Cyclic redundancy check for the given data bits.

ALGORITHM:
Step 1: Start the program.
Step 2: Accept the number of bits in the input data stream (n).
Step 3: Accept the number of bits in the divisor (d).
Step 4: Accept the input bits and the divisor bits.
Step 5: Append (d-3) '0' bits to the end of the input data bits.
Step 6: Perform binary division.
Step 7: Remove the appended '0' bits and append the remainder bits to the original data bits.
Step 8: Again perform binary division.
Step 9: If the remainder is zero then there is no error in the transmitted bits else the received bits are in error.
Step 10: Stop the program.
CODING:

```
#include<stdio.h>
#include<conio.h>
void main()
{ int
```

```
arr1[20],arr2[20],divarr[10],n,d,i,j,k,l,x,z=0,temp1[]={1,1,1,1,1,1,1},temp0[]={0,0,0,0,0,
0,0},
restarr[10],restarr1[10];
printf("Enter the number bit(s) in data\n");
scanf("%d",&n);
printf("Enter the number of bit(s) in the divisor\n");
scanf("%d",&d);
printf("Enter the data\n");
for(i=0;i<n;i++)
scanf("%d",&arr1[i]);
printf("Enter the divisor\n");
for(j=0;j<d;j++)
scanf("%d",&divarr[j]);
for(i=0;i<n;i++)
{ arr2[i]=arr1[i];
}
for(i=0;i<d-1;i++)
{ arr1[n]=0;
n++;
}
for(i=0;i<n;i++)
{
if(arr1[i]==divarr[0])
{
k=i;
for(j=0;j<d;j++)
{
if(arr1[k]==divarr[j])
{
arr1[k]=0;
k++;
} else
{
arr1[k]=1;
k++;
}
}
} else if(arr1[i]==1)
{
for(j=0;j<d;j++)
{
k=i;
if(arr1[k]==temp1[j])
{
arr1[k]=0;
k++;
} else
```

```
{
arr1[k]=1;
k++;
}
}
}
else if(arr1[i]==0)
{
for(j=0;j<d;j++)
{
k=i;
if(arr1[k]==temp0[j])
{
arr1[k]=0;
k++;
} else
{
arr1[k]=1;
k++;
}
}
}
}
// printf("the n value is %d",n);
x=n;
for(i=0;i<d-1;i++)
{
restarr[i]=arr1[x-2];
x++;
}
// printf("\n\n\nn value %d ",n);
printf("The first remainder is\n");
for(i=0;i<d-1;i++)
{
printf("%d\n",restarr[i]);
}
getch();
printf("The original value is\n");
for(i=0;i<6;i++)
{
printf("%d\n",arr2[i]);
}
x=n;
for(i=0;i<d-1;i++)
{
arr2[x-3]=restarr[i];
x++;
```

```
}
printf("The remainder bit is appended with the original data\n");
for(i=0;i<n;i++)
{
printf("%d\n",arr2[i]);
}
for(i=0;i<n;i++)
{
if(arr2[i]==divarr[0])
{
k=i;
for(j=0;j<d;j++)
{
if(arr2[k]==divarr[j])
{
arr2[k]=0;
k++;
} else
{
arr2[k]=1;
k++;
}
}
} else if(arr2[i]==1)
{
for(j=0;j<d;j++)
{
k=i;
if(arr2[k]==temp1[j])
{
arr2[k]=0;
k++;
} else
{
arr2[k]=1;
k++;
}
}
}
else if(arr2[i]==0)
{
for(j=0;j<d;j++)
{
k=i;
if(arr2[k]==temp0[j])
{
arr1[k]=0;
```

```
k++;
} else
{
arr2[k]=1;
k++;
}
}
}
for(i=0;i<d-1;i++)
{
restarr1[i]= arr2[n-3];
n++;
}
printf("The final remainder is\n");
for(i=0;i<d-1;i++)
{
printf("%d\n",restarr1[i]);
}
for(i=0;i<d-1;i++)
{
if(restarr1[i]==0)
z++;
}
if(z==d-1)
printf("There is no error in the bit\n");
getch();
}
```

OUTPUT:

```
Enter the no of bits in the dividend: 5
Enter the no of bits in the divisor: 2
Enter the dividend:
11010
Enter the divisor:
10
The first remainder is : 1
The original value is:
11010 The remainder bits appended with the original data;
```

EXPERIMENT NUMBER: 7

Objective: To Implement the TCP/IP Model.

Aim

To implement the TCP/IP Model and understand the communication process between two computers.

Objective

- To study the four layers of the TCP/IP Model.
- To understand how data is transmitted from the sender to the receiver.
- To observe the functioning of TCP/IP protocols during communication.

Requirements

- Cisco Packet Tracer
- 2 PCs
- 1 Switch
- Copper Straight-Through Cable

Theory

The **TCP/IP (Transmission Control Protocol/Internet Protocol) Model** is the standard communication model used on the Internet. It consists of four layers:

TCP/IP Communication

Application Layer
↓
Transport Layer (TCP)
↓
Internet Layer (IP)
↓
Network Access Layer
↓
Physical Network
↓
Receiver

Procedure

Step 1

Open **Cisco Packet Tracer**.

Step 2

Add the following devices:

- PC0
- PC1
- Switch 2960

Step 3

Connect the PCs to the Switch using **Copper Straight-Through Cable**.

Step 4

Assign IP Addresses.

Step 5

Open **Desktop → Command Prompt** on PC0.

Run the following command:

If configuration is correct, the output will be:

Reply from 192.168.1.20:
Packets Sent = 4
Packets Received = 4
Lost = 0

Implementation of TCP/IP Model

During the Ping operation:

1. The **Application Layer** generates the Ping request.
2. The **Transport Layer** prepares the data (ICMP uses IP directly rather than TCP/UDP for Ping).
3. The **Internet Layer** adds the source and destination IP addresses.
4. The **Network Access Layer** converts the packet into Ethernet frames and sends it through the switch.
5. The destination PC receives the packet and sends an Echo Reply.

Observation

- IP addresses were configured successfully.
- The switch established communication between the two computers.
- Ping replies were received successfully.
- The TCP/IP communication process was verified.

Result

The TCP/IP Model was successfully implemented using Cisco Packet Tracer, and communication between the computers was verified through the Ping command.

EXPERIMENT NUMBER: 8

Objective: To write a program to simulate stop and wait protocol

ALGORITHM:

Step 1: Start the program.

Step 2: Generate a random that gives the total number of frames to be transmitted.

Step 3: Transmit the first frame.

Step 4: Receive the acknowledgement for the first frame.

Step 5: Transmit the next frame

Step 6: Find the remaining frames to be sent.

Step 7: If an acknowledgement is not received for a particular frame retransmit that frame

alone again.

Step 8: Repeat the steps 5 to 7 till the number of remaining frames to be send becomes

zero.

Step 9: Stop the program.

CODING:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{ int i,j,noframes,x,x1=10,x2;
clrscr();
for(i=0;i<200;i++)
rand();
noframes=rand()/200;
i=1;
j=1;
noframes = noframes / 8;
printf("\n number of frames is %d",noframes);
getch();
while(noframes>0)
{
printf("\nsending frame %d",i);
srand(x1++);
x = rand()%10;
```

```
if(x%2 == 0)
{
for (x2=1; x2<2; x2++)
{
printf("waiting for %d seconds\n", x2);
sleep(x2);
}
printf("\nsending frame %d",i);
srand(x1++);
x = rand()%10;
}
printf("\nack for frame %d",j);
noframes-=1;
i++;
j++;
}
printf("\n end of stop and wait protocol");
getch();
}
```

OUTPUT:

No of frames is 6
Sending frame 1
Acknowledgement for frame 1
Sending frame 2
Acknowledgement for frame 2
Sending frame 3
Acknowledgement for frame 3
Sending frame 4
Acknowledgement for frame 4
Sending frame 5
Waiting for 1 second
Retransmitting frame 5
Acknowledgement for frame 5
Sending frame 6
Waiting for 1 second
Sending frame 6
Acknowledgement for frame 6
End of stop and wait protocol

EXPERIMENT NUMBER: 9

Objective: To write a program to find the Physical Address for a given IP addresses using Simulation.

ALGORITHM:

Step 1: Start the program

Step 2: Generate random numbers to get IP addresses and physical addresses

Step 3: Store the IP address and physical addresses in an array

Step 4: Randomly choose an IP address and identify the corresponding Physical address for that IP address with the help of the array.

Step 5: If a match is found then display the IP address and the corresponding physical address otherwise display that no match was found.

Step 6: Similarly choose a physical address randomly and find the corresponding IP address for that Physical address with the help of the array.

Step 7: If a match is found then display the physical address and the corresponding IP address otherwise display that no match was found.

Step 8: Stop the program.

CODING:

```
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
void main()
{
struct
{
char *ipa;
char *pha;
}ipadd[2];
inti,j,x=10;
char *temp,*temp1,*x1;
charip[2][15]={"135.237.105.128","225.22.205.221"};
charph[2][18]={"SYSTEM1","SYSTEM12"};
clrscr();
for(j=0;j<2;j++)
```

```
{
srand(x++);
for(i=0;i<4;i++)
{
itoa(rand()%256,temp,10);
printf("\n%s",temp);
strcat(ipadd[j].ipa,temp);
if(i<3)
strcat(ipadd[j].ipa,".");
}
printf("%s",itoa(j+1,x1,10));
temp1=strcat("SYSTEM",x1);
strcpy(ipadd[j].pha,temp1);
printf("\n%s\n",ipadd[j].ipa);
printf("\n%s\n",ipadd[j].pha);
}
for(j=0;j<2;j++)
printf("\n\n%s\n",ipadd[j].ipa);
for(j=0;j<2;j++)
for(i=0;i<2;i++)
if(strcmp(ipadd[i].ipa,ip[j])==0)
{
printf("\n the physical address of the given ip address %s is",ip[j]);
printf("\n\n%s\n",ph[j]);
}
getch();
for(j=0;j<2;j++)
for(i=0;i<2;i++)
if(strcmp(ipadd[i].pha,ph[j])==0)
{
printf("\n the ip address of the given physical address %s is",ph[j]);
printf("\n\n%s\n",ip[j]);
}
```

```
getch();
}
```

OUTPUT:

the physical address of the given ip address 135.237.105.128 is SYSTEM1

the physical address of the given ip address 225.22.205.221 is SYSTEM2

theip address of the given physical address SYSTEM1 is 135.237.105.128

theip address of the given physical address SYSTEM2 is 225.22.205.221

EXPERIMENT NUMBER: 10

Objective: To write a program to perform client server chat application.

ALGORITHM:

Step 1: Start the program.
Step 2: Establish the connection between the client and the server.
Step 3: Identify the Server address, family and port number.
Step 4: Create Socket to send the message
Step 5: Bind the connection
Step 6: Confirm the connection between the client and the server
Step 7: Create a buffer to send and receive the message
Step 8: Start sending the messages to and from the client and server
Step 9: Repeat the steps 4 to 8 until the connection is terminated.
Step 10: Stop the program.

CODING:

CLIENT PROGRAM:

```
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5003
int main(intargc, char *argv[])
{ intsockfd;
structsockaddr_inserv_addr;
structhostnet *server;
char buffer[4096];
sockfd = socket(AF_INET,SOCK_STREAM,0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
serv_addr.sin_port = htons(SERV_TCP_PORT);
connect(sockfd,(structsockaddr *)&serv_addr,sizeof(serv_addr));
printf("\nenter the message:\n");
bzero(buffer,4096);
printf("\nclient:");
fgets(buffer,4096,stdin);
write(sockfd,buffer,4096);
bzero(buffer,4096);
read(sockfd,buffer,4096);
printf("\nservermsg :%s",buffer);
```

```
printf("\n");
close(sockfd);
return 0;
}
```

SERVER PROGRAM:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5003
int main(intargc, char *argv[])
{ intsockfd, newsockfd, clength;
structsockaddr_inserv_addr, cli_addr;
char buffer[4096];
sockfd = socket(AF_INET,SOCK_STREAM,0);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(SERV_TCP_PORT);
bind(sockfd,(structsockaddr *)&serv_addr, sizeof(serv_addr));
listen(sockfd,5);
clength = sizeof(cli_addr);
newsockfd = accept(sockfd,(structsockaddr *)&cli_addr, &clength);
bzero(buffer,4096);
read(newsockfd,buffer,4096);
printf("\nclient message: %s",buffer);
write(newsockfd,buffer,4096);
printf("\n");
close(sockfd);
return 0;
}
```

OUTPUT:

CLIENT SIDE:

```
Enter the message:
Client:
Hai
Server message:
Hai
```

SERVER SIDE:

Client message:

Hai

Experiment No 11

Problem Statement

To connect two personal computer with straight thru and cross over twisted pair

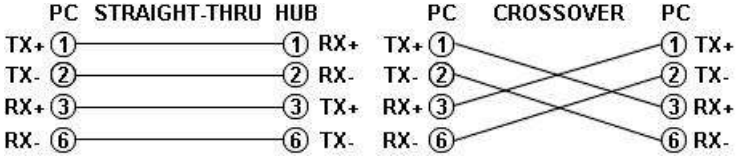
Theory

A **crossover cable** connects two devices of the same type, for example DTE-DTE or DCE-DCE, usually connected asymmetrically (DTE-DCE), and by a modified cable called a crosslink. Such distinction of devices was introduced by IBM.

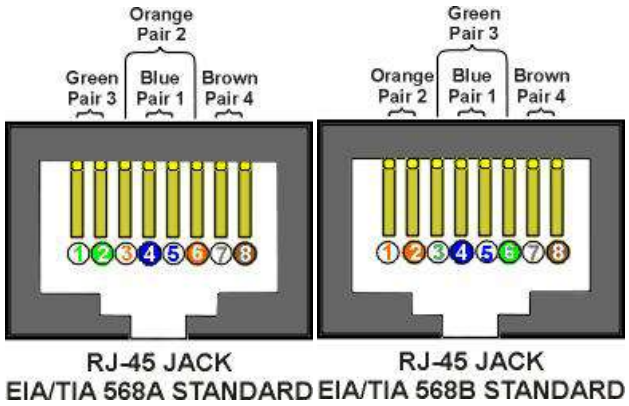


The crossing wires in a cable or in a connector adaptor allows:

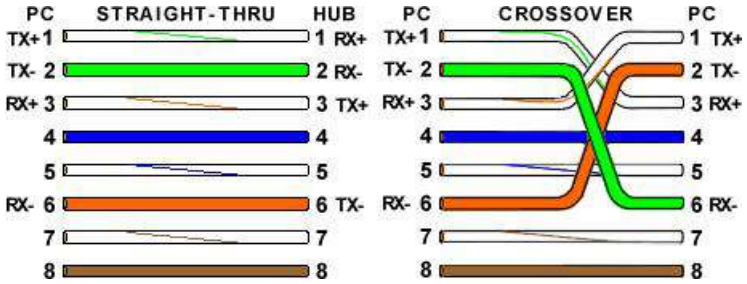
- connecting two devices directly, output of one to input of the other,
- letting two terminal (DTE) devices communicate without an interconnecting hub knot, i.e. PCs,
- linking two or more hubs, switches or routers (DCE) together, possibly to work as one wider device.
- Let's start with simple pin-out diagrams of the two types of UTP Ethernet cables and watch how committees can make a can of worms out of them. Here are the diagrams:



- Note that the TX (transmitter) pins are connected to corresponding RX (receiver) pins, plus to plus and minus to minus. And that you must use a crossover cable to connect units with identical interfaces. If you use a straight-through cable, one of the two units must, in effect, perform the cross-over function.
- Two wire color-code standards apply: EIA/TIA 568A and EIA/TIA 568B. The codes are commonly depicted with RJ-45 jacks as follows (the view is from the front of the jacks):

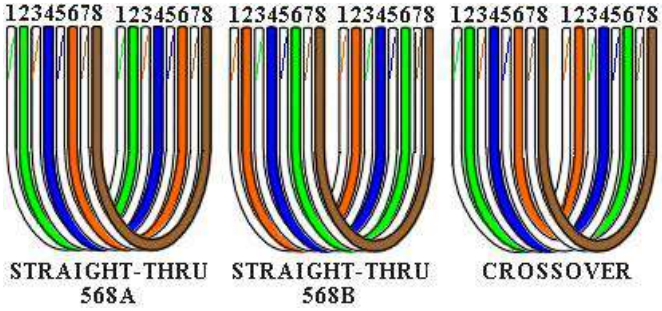


- If we apply the 568A color code and show all eight wires, our pin-out looks like this:



- Note that pins 4, 5, 7, and 8 and the blue and brown pairs are not used in either standard. Quite contrary to what you may read elsewhere, these pins and wires are not used or required to implement 100BASE-TX duplexing--they are just plain wasted.
- However, the actual cables are not physically that simple. In the diagrams, the orange pair of wires are not adjacent. The blue pair is upside-down. The right ends match RJ-45 jacks and the left ends do not. If, for example, we invert the left side of the 568A "straight"-thru cable to match a 568A jack--put one 180° twist in the entire cable from end-to-end--and twist together and rearrange the appropriate pairs, we get the following can-of-worms:
- This further emphasizes, I hope, the importance of the word "twist" in making network cables which will work. You cannot use an flat-untwisted telephone cable for a network cable. Furthermore, you must use a pair of twisted wires to connect a set of transmitter pins to their corresponding receiver pins. You cannot use a wire from one pair and another wire from a different pair.
- Keeping the above principles in mind, we can simplify the diagram for a 568A straight-thru cable by untwisting the wires, except the 180° twist in the entire cable, and bending the ends upward. Likewise, if we exchange the green and orange pairs in the 568A diagram we will get a simplified diagram for a 568B straight-thru cable. If we cross the green and orange pairs in

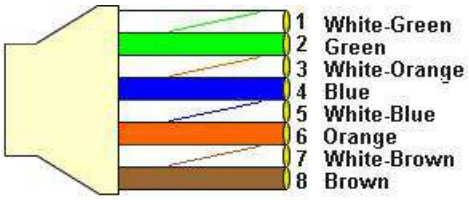
the 568A diagram we will arrive at a simplified diagram for a crossover cable. All three are shown below.



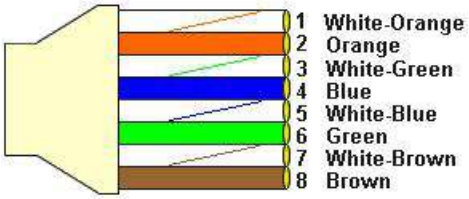
There are only two unique cable ends in the preceding diagrams. They correspond to the 568A and 568B RJ-45 jacks and are shown to the right.
Again, the wires with colored backgrounds may have white stripes and may be denoted that way in diagrams found elsewhere. For example, the green wire may be labeled Green-White--I don't bother. The background color is always specified first.

Now, all you need to remember, to properly configure the cables, are the diagrams for the two cable ends and the following rules:

- **A straight-thru cable has identical ends.**
- **A crossover cable has different ends.**



568A CABLE END



568B CABLE END

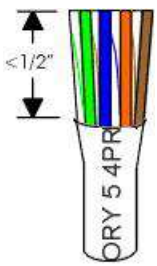
It makes no functional difference which standard you use for a straight-thru cable. You can start a crossover cable with either standard as long as the other end is the other standard. It makes no functional difference which end is which. Despite what you may have read elsewhere, a 568A patch cable will work in a network with 568B wiring and 568B patch cable will work in a 568A network. The electrons couldn't care less.



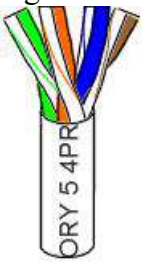
Procedure:

1. Pull the cable off the reel to the desired length and cut. I have a box of cable at one end of my shop and a mark on the floor 10' away. For cable lengths which are a fraction of ten feet, I eye-ball the length as I pull the cable out of the box (also, my feet are about one foot long). For longer cables, I pull it out to the ten foot mark and go back to the box and pull the remaining fraction or another ten feet. If you are pulling cables through walls, a hole in the floor, etc., it easier to attach the RJ-45 plugs after the cable is pulled. The total length of wire segments between a PC and a hub or between two PC's cannot exceed 100 Meters (328 feet or about the length of a football field) for 100BASE-TX (and 10BASE-T).





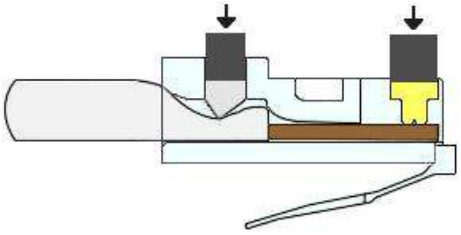
2. Strip one end of the cable with the stripper or a knife and diags. If you are using the stripper, place the cable in the groove on the blade (left) side of the stripper and align the end of the cable with the right side of the stripper. This is about right to strip a little over 1/2" of the jacket off the cable. Turn the stripper about one turn or so. If you turn it much more, you will probably nick the wires. The idea is to score the outer jacket, but not go all the way through. Once scored, you should be able to twist the end of the jacket loose and pull it off with one hand while holding the rest of the cable with the other. If you are using a knife and diags, carefully slit the cable

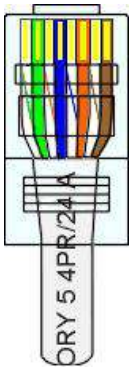


for about an inch or so and neatly trim around the circumference of the cable with the diags to remove the jacket.

3. Inspect the wires for nicks. Cut off the end and start over if you see any. You may have to adjust the blade with the screw at the front stripper. Cable diameters and jacket thicknesses vary.







4. Spread and arrange the pairs roughly in the order of the desired cable end.
5. Untwist the pairs and arrange the wires in the order of the desired cable end. Flatten the end between your thumb and forefinger. Trim the ends of the wires so they are even with one another. **It is very important that the unstripped (untwisted) end be slightly less than 1/2" long.** If it is longer than 1/2" it will be out-of-spec and susceptible to crosstalk. If it less than slightly less than 1/2" it will not be properly clinched when RJ-45 plug is crimped on.. Flatten again. There should be little or no space between the wires.
6. Hold the RJ-45 plug with the clip facing down or away from you. Push the wire firmly into the plug. **Now**, **inspect the darn thing... before crimping and wasting the plug!** Looking through the bottom of the plug, the wire on the far left side will have a white background. The wires should alternate light and dark from left to right. The furthest right wire is brown. The wires should all end evenly at the front of the plug. The jacket should end just about where you see it in the diagram--right on the line. Aren't you glad you didn't crimp the plug?
7. Hold the wire near the RJ-45 plug with the clip down and firmly push it into the left side of the front of the crimper (it will only go in one way). Hold the wire in place squeeze the crimper handles quite firmly. This is what will happen:
(Crimp it once.) The crimper pushes two plungers down on the RJ-45 plug. One forces what amounts to a cleverly designed plastic plug/wedge onto the cable jacket and very firmly clinches it. The other seats the "pins," each with two teeth at its end, through the insulation and into the conductors of their respective wires.
8. Test the crimp... If done properly an average person will not be able to pull the plug off the cable with his or her bare hands. And that quite simply, besides lower cost, is the primary advantage of twisted-pair cables over the older thinwire, coaxial cables. In fact, I would say the RJ-45 and ease of its installation is the main reason coaxial cable is no longer widely used for small Ethernets. But, don't pull that hard on the plug. It could stretch the cable and change its characteristics. Look at the side of the plug and see if it looks like the diagram and give it a fairly firm tug to make sure it is crimped well.
9. Prepare the other end of the cable so it has the desired end and crimp.
10. If both ends of the cable are within reach, hold them next to each other and with RJ-45 clips facing away. Look through the bottom of the plugs. If the plugs are wired correctly, and they are identical, it is a straight-thru cable. If they are wired correctly and they are different, it is a crossover cable.
11. If you have an operational network, test the cable. Copy some large files.

12. If the cable doesn't work, inspect the ends again and make sure you have the right cable and that it is plugged into the correct units for the type of cable. Try power-cycling (cold booting) the involved computers.
13. If you have many straight-thru cables and a crossover cable in your system, you should consider labeling the crossover cable or using a different colored cable for the crossover cable so you don't mix them up. I do not recommend implementing the crossover function, as recommended elsewhere, with two RJ-45 jacks, appropriately wired back to back, and two straight-thru cables. This method costs noticeably more, introduces more than the necessary number of components and connections, increases the complexity and time of assembly, and decreases reliability.

Precautions:

1. Try to avoid running cables parallel to power cables.
2. Do not bend cables to less than four times the diameter of the cable.
3. If you bundle a group of cables together with cable ties (zip ties), do not over-cinch them. It's okay to snug them together firmly; but don't tighten them so much that you deform the cables.
4. Keep cables away from devices which can introduce noise into them. Here's a short list: copy machines, electric heaters, speakers, printers, TV sets, fluorescent lights, copiers, welding machines, microwave ovens, telephones, fans, elevators, motors, electric ovens, dryers, washing machines, and shop equipment.
5. Avoid stretching UTP cables (tension when pulling cables should not exceed 25 LBS).
6. **Do not run UTP cable outside of a building. It presents a very dangerous lightning hazard!**
7. Do not use a stapler to secure UTP cables. Use telephone wire/RG-6 coaxial wire hangers which are available at most hardware stores.

Experiment No 12

Problem Statement

To implement LZW compression algorithm in C.

Theory

The original Lempel Ziv approach to data compression was first published in in 1977, followed by an alternate approach in 1978. Terry Welch's refinements to the 1978 algorithm were published in 1984. The algorithm is surprisingly simple. In a nutshell, LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.

The code that the LZW algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds. The sample program runs as shown with 12 bit codes. This means codes 0-255 refer to individual bytes, while codes 256-4095 refer to substrings.

The LZW compression algorithm in its simplest form is shown in Figure 1. A quick examination of the algorithm shows that LZW is always trying to output codes for strings that are already known. And each time a new code is output, a new string is added to the string table.

1.3 FLOW CHART / ALGORITHM:

```
1.  STRING = get input character
2.  WHILE there are still input characters DO
3.      CHARACTER = get input character
4.      IF STRING+CHARACTER is in the string table then
5.          STRING = STRING+character
6.      ELSE
7.          output the code for STRING
8.          add STRING+CHARACTER to the string table
9.          STRING = CHARACTER
10.     END of IF
11. END of WHILE
12. output the code for STRING
```

A sample string used to demonstrate the algorithm is shown in Figure 2. The input string is a short list of English words separated by the '/' character. Stepping through the start of the algorithm for this string, you can see that the first pass through the loop, a check is performed to see if the string "/W" is in the table. Since it isn't, the code for '/' is output, and the string "/W" is added to the table. Since we have 256 characters already defined for codes 0-255, the first string definition can be assigned to code 256. After the third letter, 'E', has been read in, the second string code, "WE" is added to the table, and the code for letter 'W' is output. This continues until in the second word, the characters '/' and 'W' are read in, matching string number 256. In this case, the code 256 is output, and a three character string is added to the string table. The process continues until the string is exhausted and all of the codes have been output.

Input String = /WED/WE/WEE/WEB/WET			
Character Input	Code Output	New code value	New String
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

The sample output for the string is shown in Figure 2 along with the resulting string table. As can be seen, the string table fills up rapidly, since a new string is added to the table each time a code is output. In this highly redundant input, 5 code substitutions were output, along with 7 characters. If we were using 9 bit codes for output, the 19 character input string would be reduced to a 13.5 byte output string. Of course, this example was carefully chosen to demonstrate code substitution. In real world examples, compression usually doesn't begin until a sizable table has been built, usually after at least one hundred or so bytes have been read in.

Decompression

The companion algorithm for compression is the decompression algorithm. It needs to be able to take the stream of codes output from the compression algorithm, and use them to exactly recreate the input stream. One reason for the efficiency of the LZW algorithm is that it does not need to pass the string table to the decompression code. The table can be built exactly as it was during compression, using the input stream as data. This is possible because the compression algorithm always outputs the STRING and CHARACTER components of a code before it uses it in the output stream. This means that the compressed data is not burdened with carrying a large string translation table.

CODE:

1. Read OLD_CODE
2. output OLD_CODE
3. WHILE there are still input characters DO
4. Read NEW_CODE
5. STRING = get translation of NEW_CODE
6. output STRING
7. CHARACTER = first character in STRING
8. add OLD_CODE + CHARACTER to the translation table
9. OLD_CODE = NEW_CODE
10. END of WHILE

The algorithm is shown in Figure 3. Just like the compression algorithm, it adds a new string to the string table each time it reads in a new code. All it needs to do in addition to that is translate each incoming code into a string and send it to the output.

Figure shows the output of the algorithm given the input created by the compression earlier in the article. The important thing to note is that the string table ends up looking exactly like the table built up during compression. The output string is identical to the input string from the compression algorithm. Note that the first 256 codes are already defined to translate to single character strings, just like in the compression code.

Input Codes: / W E D 256 E 260 261 257 B 260 T				
Input/ NEW_CODE	OLD_CODE	STRING/ Output	CHARACTER	New table entry
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

The Catch

Unfortunately, the nice simple decompression algorithm shown in Figure 4 is just a little *too* simple. There is a single exception case in the LZW compression algorithm that causes some trouble to the decompression side. If there is a string consisting of a (STRING,CHARACTER) pair already defined in the table, and the input stream then sees a sequence of STRING, CHARACTER, STRING, CHARACTER, STRING, the compression algorithm will output a code before the decompressor gets a chance to define it.
A simple example will illustrate the point. Imagine the the string JOEYN is defined in the table as code 300. Later on, the sequence JOEYNJOEYNJOEY occurs in the table. The compression output will look like that shown in Figure 5.

Input String: ...JOEYNJOEYNJOEY		
Character Input	New Code/String	Code Output
JOEYN	300 = JOEYN	288 (JOEY)
A	301 = NA	N
.	.	.
.	.	.
.	.	.
JOEYNJ	400 = JOEYNJ	300 (JOEYN)
JOEYNJO	401 = JOEYNJO	400 (???)

When the decompression algorithm sees this input stream, it first decodes the code 300, and outputs the JOEYN string. After doing the output, it will add the definition for code 399 to the table, whatever

that may be. It then reads the next input code, 400, and finds that it is not in the table. This is a problem, what do we do?
Fortunately, this is the only case where the decompression algorithm will encounter an undefined code. Since it is in fact the only case, we can add an exception handler to the algorithm. The modified algorithm just looks for the special case of an undefined code, and handles it. In the example in Figure 5, the decompression routine sees a code of 400, which is undefined. Since it is undefined, it translates the value of OLD_CODE, which is code 300. It then adds the CHARACTER value, which is 'J', to the string. This results in the correct translation of code 400 to string "JOEYNJ".

CODE:

1. Read OLD_CODE
2. output OLD_CODE
3. CHARACTER = OLD_CODE
4. WHILE there are still input characters DO
5. Read NEW_CODE
6. IF NEW_CODE is not in the translation table THEN
7. STRING = get translation of OLD_CODE
8. STRING = STRING+CHARACTER
9. ELSE
10. STRING = get translation of NEW_CODE
11. END of IF
12. output STRING
13. CHARACTER = first character in STRING
14. add OLD_CODE + CHARACTER to the translation table
15. OLD_CODE = NEW_CODE
16. END of WHILE

1.4 CODING

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

#define BITS 12                 /* Setting the number of bits to 12, 13*/
#define HASHING_SHIFT (BITS-8)    /* or 14 affects several constants.    */
#define MAX_VALUE (1 << BITS) - 1 /* Note that MS-DOS machines need to   */
#define MAX_CODE MAX_VALUE - 1    /* compile their code in large model if*/
                          /* 14 bits are selected.              */
#if BITS == 14
  #define TABLE_SIZE 18041       /* The string table size needs to be a */
#endif                          /* prime number that is somewhat larger*/
#if BITS == 13                  /* than 2**BITS.                       */
  #define TABLE_SIZE 9029
#endif
#if BITS <= 12
  #define TABLE_SIZE 5021
#endif

void *malloc();

int *code_value;                /* This is the code value array        */
unsignedint *prefix_code;       /* This array holds the prefix codes   */
unsigned char *append_character;  /* This array holds the appended chars */
unsigned char decode_stack[4000]; /* This array holds the decoded string */
void compress(FILE *input,FILE *output);
void expand(FILE *input,FILE *output);
intfind_match(inthash_prefix,unsignedinthash_character);
voidoutput_code(FILE *output,unsignedint code);
unsignedintinput_code(FILE *input);
```

```
unsigned char *decode_string(unsigned char *buffer,unsignedint code);

main(intargc, char *argv[])
{
FILE *input_file;
FILE *output_file;
FILE *lzw_file;
charinput_file_name[81];

code_value=(int*)malloc(TABLE_SIZE*sizeof(int));
prefix_code=(unsigned int *)malloc(TABLE_SIZE*sizeof(unsigned int));
append_character=(unsigned char *)malloc(TABLE_SIZE*sizeof(unsigned char));
if (code_value==NULL || prefix_code==NULL || append_character==NULL)
  {
printf("Fatal error allocating table space!\n");
exit(-1);
  }
if (argc>1)
strcpy(input_file_name,argv[1]);
else
  {
printf("Input file name? ");
scanf("%s",input_file_name);
  }
input_file=fopen(input_file_name,"rb");
lzw_file=fopen("test.lzw","wb");
if (input_file==NULL || lzw_file==NULL)
  {
printf("Fatal error opening files.\n");
exit(-1);
  };
compress(input_file,lzw_file);
fclose(input_file);
fclose(lzw_file);
free(code_value);
lzw_file=fopen("test.lzw","rb");
output_file=fopen("test.out","wb");
if (lzw_file==NULL || output_file==NULL)
  {

printf("Fatal error opening files.\n");
exit(-2);
  };
expand(lzw_file,output_file);
fclose(lzw_file);
fclose(output_file);

free(prefix_code);
```

```
free(append_character);
}


void compress(FILE *input,FILE *output)
{
unsignedintnext_code;
unsignedint character;
unsignedintstring_code;
unsignedint index;
inti;

next_code=256;            /* Next code is the next available string code*/
for (i=0;i<TABLE_SIZE;i++)  /* Clear out the string table before starting */
code_value[i]=-1;

i=0;
printf("Compressing...\n");
string_code=getc(input);    /* Get the first code                    */
while ((character=getc(input)) != (unsigned)EOF)
  {
if (++i==1000)                    /* Print a * every 1000    */
   {                             /* input characters.  This */
i=0;                            /* is just a pacifier.     */
printf("*");
   }
index=find_match(string_code,character);/* See if the string is in */
if (code_value[index] != -1)          /* the table.  If it is,   */
string_code=code_value[index];       /* get the code value.  If */
else                                 /* the string is not in the*/
   {                                 /* table, try to add it.   */
if (next_code<= MAX_CODE)
     {
code_value[index]=next_code++;
prefix_code[index]=string_code;
append_character[index]=character;
     }
output_code(output,string_code);  /* When a string is found  */
string_code=character;            /* that is not in the table*/
   }                              /* I output the last string*/
 }                                /* after adding the new one*/

output_code(output,string_code); /* Output the last code              */
output_code(output,MAX_VALUE);   /* Output the end of buffer code     */
output_code(output,0);           /* This code flushes the output buffer*/
printf("\n");
}
intfind_match(inthash_prefix,unsignedinthash_character)
```

```
{
int index;
int offset;

index = (hash_character<< HASHING_SHIFT) ^ hash_prefix;
if (index == 0)
offset = 1;
else
offset = TABLE_SIZE - index;
while (1)
  {
if (code_value[index] == -1)
return(index);
if (prefix_code[index] == hash_prefix&&
append_character[index] == hash_character)
return(index);
index -= offset;
if (index < 0)
index += TABLE_SIZE;
  }
}


void expand(FILE *input,FILE *output)
{
unsignedintnext_code;
unsignedintnew_code;
unsignedintold_code;
int character;
int counter;
unsigned char *string;

next_code=256;          /* This is the next available code to define */
counter=0;              /* Counter is used as a pacifier.            */
printf("Expanding...\n");

old_code=input_code(input);  /* Read in the first code, initialize the */
character=old_code;          /* character variable, and send the first */
putc(old_code,output);       /* code to the output file                */
while ((new_code=input_code(input)) != (MAX_VALUE))
  {
if (++counter==1000)   /* This section of code prints out     */
    {                  /* an asterisk every 1000 characters   */

counter=0;          /* It is just a pacifier.              */
printf("*");
    }
if (new_code>=next_code)
```

```
    {
     *decode_stack=character;
string=decode_string(decode_stack+1,old_code);
    }
else
string=decode_string(decode_stack,new_code);
character=*string;
while (string >= decode_stack)
putc(*string--,output);
if (next_code<= MAX_CODE)
    {
prefix_code[next_code]=old_code;
append_character[next_code]=character;
next_code++;
    }
old_code=new_code;
  }
printf("\n");
}

unsigned char *decode_string(unsigned char *buffer,unsignedint code)
{
inti;

i=0;
while (code > 255)
{
    *buffer++ = append_character[code];
code=prefix_code[code];
if (i++>=MAX_CODE)
    {
printf("Fatal error during code expansion.\n");
exit(-3);
    }
  }
  *buffer=code;
return(buffer);
}

unsignedintinput_code(FILE *input)
{
unsignedintreturn_value;
staticintinput_bit_count=0;
static unsigned long input_bit_buffer=0L;


while (input_bit_count<= 24)
  {
```

```
input_bit_buffer |=
      (unsigned long) getc(input) << (24-input_bit_count);
input_bit_count += 8;
  }
return_value=input_bit_buffer>> (32-BITS);
input_bit_buffer<<= BITS;
input_bit_count -= BITS;
return(return_value);
}

voidoutput_code(FILE *output,unsignedint code)
{
staticintoutput_bit_count=0;
static unsigned long output_bit_buffer=0L;

output_bit_buffer |= (unsigned long) code << (32-BITS-output_bit_count);
output_bit_count += BITS;
while (output_bit_count>= 8)
  {
putc(output_bit_buffer>> 24,output);
output_bit_buffer<<= 8;
output_bit_count -= 8;
  }
}
```

1.5 OUTPUT:

Input String = /WED/WE/WEE/WEB/WET			
Character Input	Code Output	New code value	New String
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE

/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

The Compression Process

Input Codes: / W E D 256 E 260 261 257 B 260 T				
Input/ NEW_CODE	OLD_CODE	STRING/ Output	CHARACTER	New table entry
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

The Decompression Process