GR20 Regulations

III B.Tech II Semester

COMPUTER NETWORKS AND SECURITY LAB

(GR20A3150)

Department of AIML Engineering

(Computer Science and Business Systems)

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

SYLLABUS

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

COMPUTER NETWORKS AND SECURITY LAB

Course Code: GR20A3150 **L/T/P/C :0/ 0/ 1/ 1**

III Year II Semester

Course Objectives:

1. To learn various networking technologies
2. To appropriately use network configuration commands
3. To learn the routing techniques
4. To learn different encryption techniques
5. To learn client/server models using sockets

Course Outcomes:

1. Understand the various networking technologies
2. Use file sharing and routing techniques
3. Implement Dijkstra 's algorithm for routing.
4. Implement DES,AES and RSA Algorithms
5. Implement socket programming for client/server models.

LIST OF EXPERIMENTS:

TASK 1:

Study of Different types of cables and network IP

TASK 2:

Study of following network devices in detail

a) Repeater b)Hub c)Bridge d)Router e)Gateway f)Switch

TASK 3:

Study and practice the basic network configuration commands

TASK 4:

Implement on a data set of characters the three CRC polynomials

TASK 5:

Simulate framing methods such as character stuffing and bit stuffing.

TASK 6:

Implement Dijkstra 's algorithm to compute the Shortest path through a graph.

TASK 7:

Implement DES Encryption and Decryption

TASK 8:

Implement the AES Encryption and decryption

TASK 9:

Implement RSA Encryption Algorithm

TASK 10:

Study of Socket Programming and Client – Server model.

TASK 11:

Write a socket program (using c) for interaction between server and client processes using Unix Domain sockets.

TASK 12:

Write a socket program (using c) for interaction between server and client processes using Internet Domain sockets.

TEXTBOOKS:

1. Data and Computer Communication, William Stallings.
2. Unix System Programming using C++, T.Chan, PHI.

REFERENCE BOOKS:

1. Data Communications and Networking – Behrouz A. Forouzan, 4thEdition TMH, 2006
2. Information Security: Principles and Practice, M. Stamp

INDEX

TASK 1

Aim: Analysis about Different types of cables and network IP

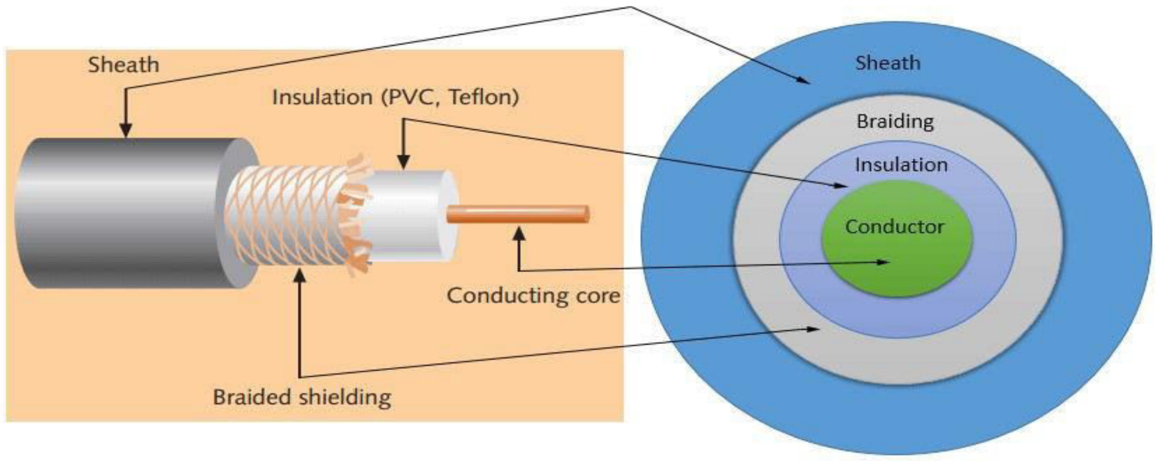
Network Cable Types and Specifications

This experiment explains the types of network cables used in computer networks in detail. Learn the specifications, standards, and features of the coaxial cable, twisted-pair cable, and the fiber-optical cable.

To connect two or more computers or networking devices in a network, network cables are used. There are three types of network cables; coaxial, twisted-pair, and fiber-optic.

Coaxial cable

This cable contains a conductor, insulator, braiding, and sheath. The sheath covers the braiding, braiding covers the insulation, and the insulation covers the conductor.



Components of Coaxial cable

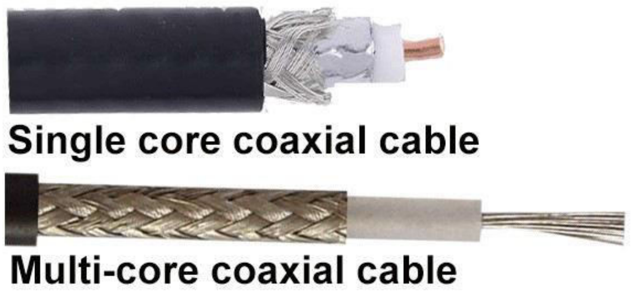
*S*heath
This is the outer layer of the coaxial cable. It protects the cable from physical damage.

Braided-shield
This shield protects signals from external interference and noise. This shield is built from the same metal that is used to build the core.

Insulation
Insulation protects the core. It also keeps the core separate from the braided-shield. Since both the core and the braided-shield use the same metal, without this layer, they will touch each other and create a short- circuit in the wire.

Conductor

The conductor carries electromagnetic signals. Based on conductor a coaxial cable can be categorized into two types; ***single-core coaxial cable and multi-core coaxial cable.***
A single-core coaxial cable uses a single central metal (usually copper) conductor,
while ***a multi-core coaxial cable*** uses multiple thin strands of metal wires. The following image shows both types of cable.



Coaxial cables in computer networks

The coaxial cables were not primarily developed for the computer network. These cables were developed for general purposes. They were in use even before computer networks came into existence. They are still used even their use in computer networks has been completely discontinued.
At the beginning of computer networking, when there were no dedicated media cables available for computer networks, network administrators began using coaxial cables to build computer networks. Because of low-cost and long durability, coaxial cables were used in computer networking for nearly two decades (80s and 90s).
Coaxial cables are no longer used to build any type of computer network.

Specifications of coaxial cables

Coaxial cables have been in use for the last four decades. During these years, based on several factors such as the thickness of the sheath, the metal of the conductor, and the material used in insulation, hundreds of specifications have been created to specify the characteristics of coaxial cables.

From these specifications, only a few were used in computer networks. The following table lists them.

Type	Ohms	AWG	Conductor	Description
RG-6	75	18	Solid copper	Used in cable network to provide cable Internet service and cable TV over long distances.
RG-8	50	10	Solid copper	Used in the earliest computer networks. This cable was used as the backbone-cable in the bus topology. In Ethernet standards, this cable is documented as the 10base5 Thicknet cable.
RG-58	50	24	Several thin strands of copper	This cable is thinner, easier to handle and install than the RG-8 cable. This cable was used to connect a system with the backbone-cable. In Ethernet standards, this cable is documented as the 10base2 Thinnet cable.
RG-59	75	20 -22	Solid copper	Used in cable networks to provide short-distance service.

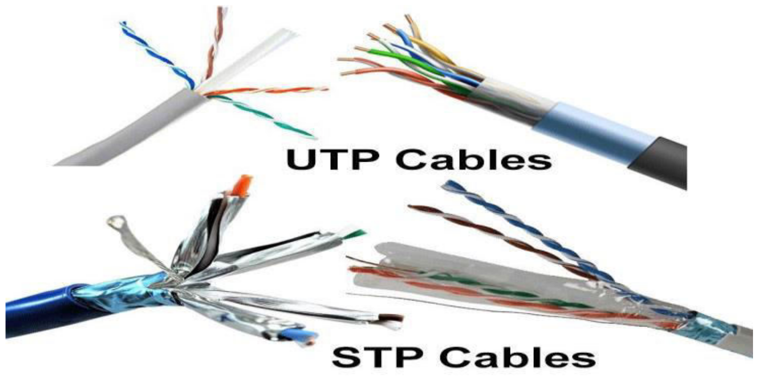
Coaxial cable uses RG rating to measure the materials used in shielding and conducting cores. RG stands for the Radio Guide. Coaxial cable mainly uses radio frequencies in transmission. Impedance is the resistance that controls the signals. It is expressed in the **ohms**.
AWG stands for American Wire Gauge. It is used to measure the size of the core. The larger the AWG size, the smaller the diameter of the core wire.

Twisted-pair cables

Usually, there are four pairs. Each pair has one solid color and one stripped color wire. Solid colors are blue, brown, green and orange. In stripped color, the solid color is mixed with the white color. Based on how pairs are stripped in the plastic sheath, there are two types of twisted-pair cable; UTP and STP. In the UTP (Unshielded twisted-pair) cable, all pairs are wrapped in a single plastic sheath. In the STP (Shielded twisted-pair) cable, each pair is wrapped with an additional metal shield, then all pairs are wrapped in a single outer plastic sheath.

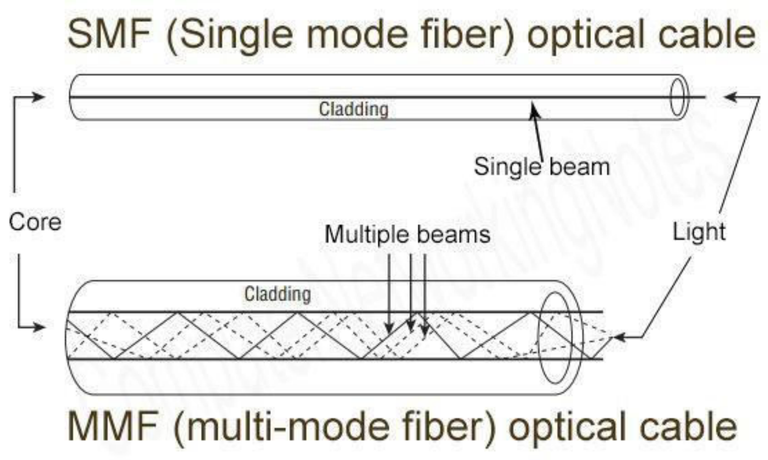
Similarities and differences between STP and UTP cables

STP	UTP
can transmit data at 10Mbps, 100Mbps, 1Gbps, and 10Gbps.	can transmit data at 10Mbps, 100Mbps, 1Gbps, and 10Gbps.
the STP cable contains more materials,	the UTP cable contains less materials,
it is more expensive than the UTP cable	it is less expensive than the STP cable
Both cables use the same RJ-45 (registered jack) modular connectors.	Both cables use the same RJ-45 (registered jack) modular connectors
The STP provides more noise and EMI resistant than the UTP cable.	The UTP provides less noise and EMI resistant than the STP cable.
The maximum segment length for both cables is 100 meters or 328 feet.	The maximum segment length for both cables is 100 meters or 328 feet.
Both cables can accommodate a maximum of 1024 nodes in each segment.	Both cables can accommodate a maximum of 1024 nodes in each segment.



Fiber optic cable
This cable consists of core, cladding, buffer, and jacket. The core is made from the thin strands of glass or plastic that can carry data over the long distance. The core is wrapped in the cladding; the cladding is wrapped in the buffer, and the buffer is wrapped in the jacket.
Core carries the data signals in the form of the light. Cladding reflects light back to the core. Buffer protects the light from leaking. The jacket protects the cable from physical damage. Fiber optic cable is

completely immune to EMI and RFI. This cable can transmit data over a long distance at the highest speed. It can transmit data up to 40 kilometers at the speed of 100Gbps.
Fiber optic uses light to send data. It reflects light from one endpoint to another. Based on how many beams of light are transmitted at a given time, there are two types of fiber optical cable; SMF and MMF.


SMF (Single mode fiber) optical cable
Cladding
Single beam
Core
Multiple beams
Light
Cladding
MMF (multi-mode fiber) optical cable

SMF (Single-mode fiber) optical cable

This cable carries only a single beam of light. This is more reliable and supports much higher bandwidth and longer distances than the MMF cable. This cable uses a laser as the light source and transmits 1300 or 1550 nano- meter wavelengths of light.

MMF (multi-mode fiber) optical cable

This cable carries multiple beams of light. Because of multiple beams, this cable carries much more data than the SMF cable. This cable is used in shorter distances. This cable uses an LED as the light source and transmits 850 or 1300 nano-meter wavelengths of light.
That's all for this tutorial. In the next part of this article, we will understand the types of connectors that are used to connect cables with networking devices. If you like this tutorial, please don't forget to share it with friends through your favorite social channel.

Fiber Optic Cabling Solutions

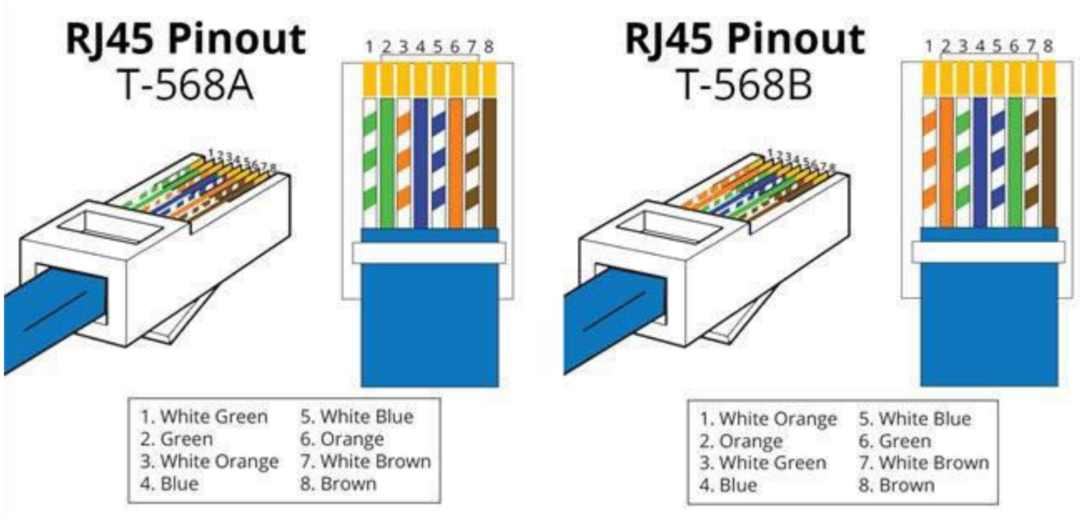
The largest solutions of pre-terminated fiber optics, including multimode and single-mode patch cords, MTP/MPO fiber trunks and harnesses, plug-n-play modules/cassettes and fiber enclosures.

Difference of Straight through and Crossover Cable

Ethernet cables can be wired as straight through or crossover. The straight through is the most common type and is used to connect computers to hubs or switches. They are most likely what you will find when you go to your local computer store and buy a patch cable. Crossover Ethernet cable is more commonly used to connect a computer to a computer and may be a little harder to find since they aren't used nearly as much as straight through Ethernet cable. Then, what's the difference between straight through vs crossover cable? Read through this post to find the answer.

T568A And T568B Wiring Standard Basis

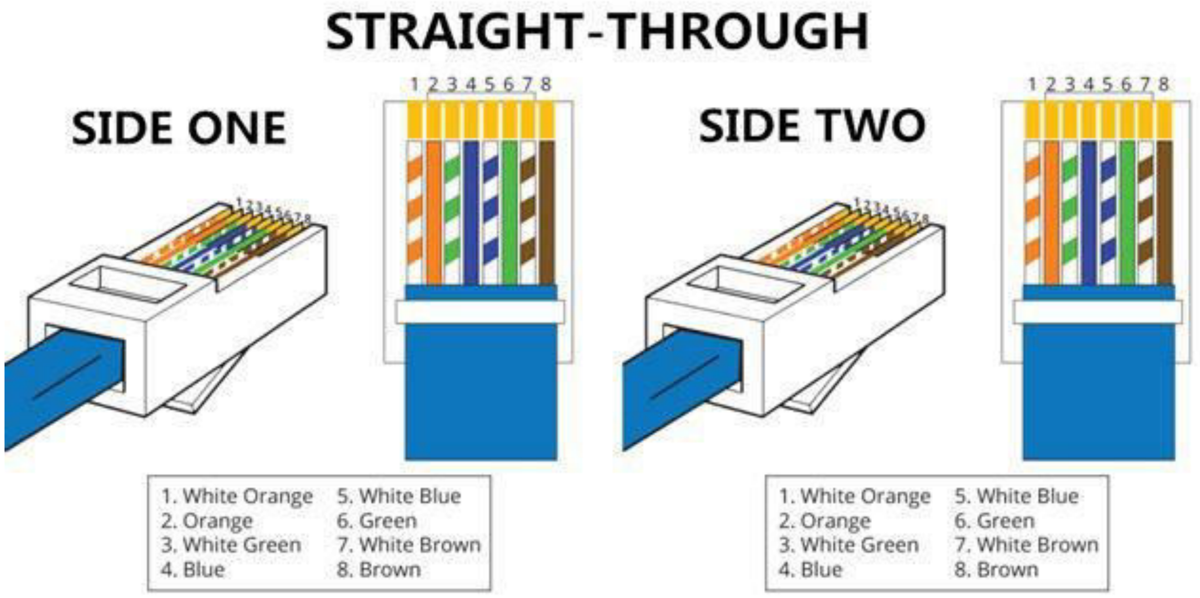
A RJ45 connector is a modular 8 position, 8 pin connector used for terminating Cat5e patch cable or Cat6 cable. A pin out is a specific arrangement of wires that dictate how the connector is terminated. There are two standards recognized by ANSI, TIA and EIA for wiring Ethernet cables. The first is the T568A wiring standard and the second is T568B. T568B has surpassed 568A and is seen as the default wiring scheme for twisted pair structured cabling. If you are unsure of which to use, choose 568B.



Straight Through vs Crossover Cable

What Is Straight Through Cable?

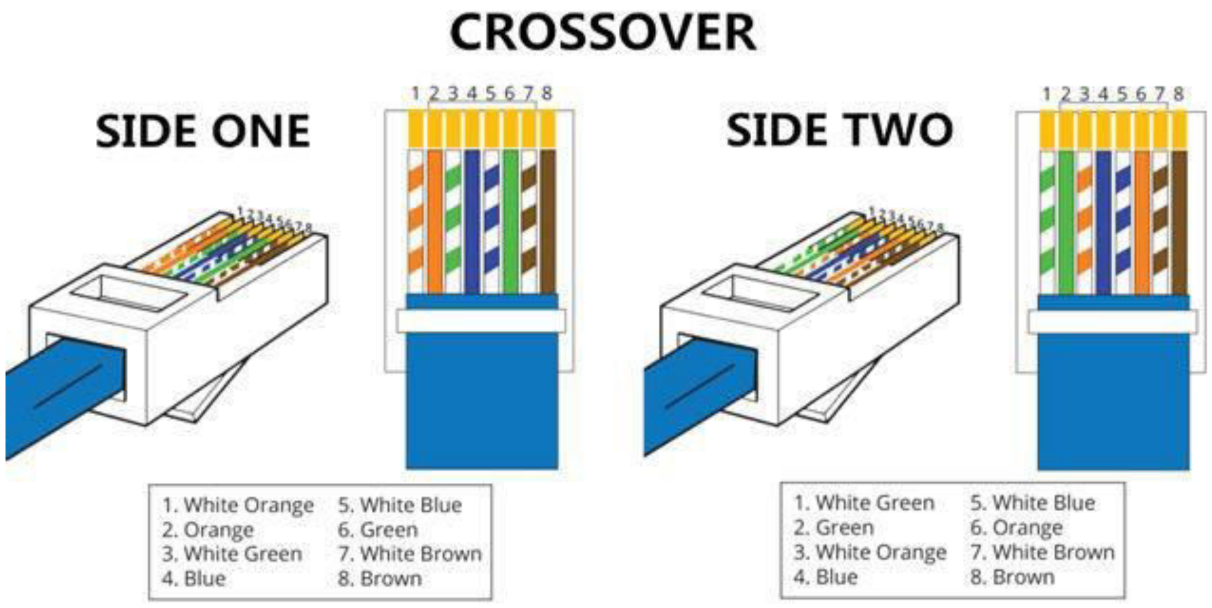
A straight through cable is a type of twisted pair cable that is used in local area networks to connect a computer to a network hub such as a router. This type of cable is also sometimes called a patch cable and is an alternative to wireless connections where one or more computers access a router through wireless signal. On a straight through cable, the wired pins match. Straight through cable use one wiring standard: both ends use T568A wiring standard or both ends use T568B wiring standard. The following figure shows a straight through cable of which both ends are wired as the T568B standard.



“A straight through cable of which both ends”

What Is Crossover Cable?

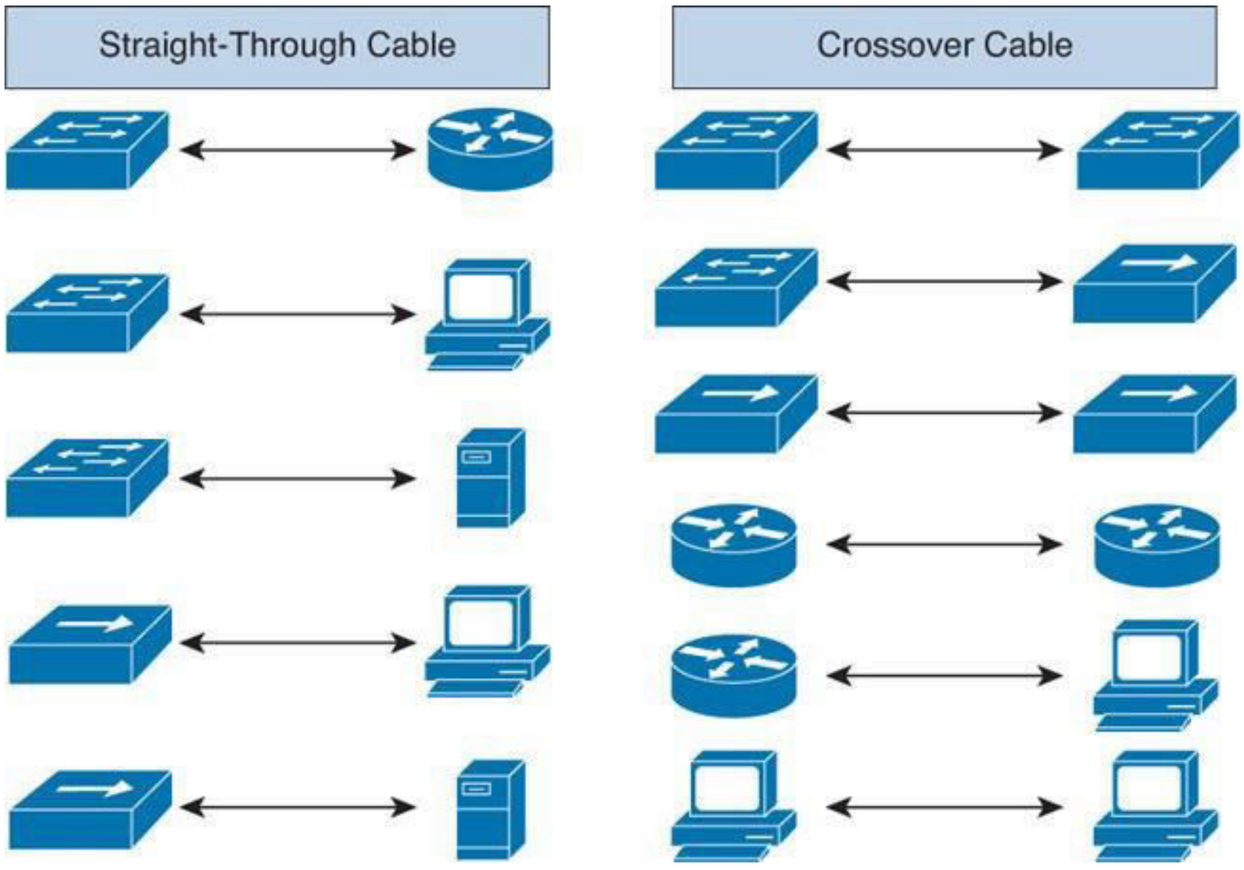
A crossover Ethernet cable is a type of Ethernet cable used to connect computing devices together directly. Unlike straight through cable, the RJ45 crossover cable uses two different wiring standards: one end uses the T568A wiring standard and the other end uses the T568B wiring standard. The internal wiring of Ethernet crossover cables reverses the transmit and receive signals. It is most often used to connect two devices of the same type: e.g. two computers (via network interface controller) or two switches to each other.



“A Crossover cable of which both ends”

Straight through vs Crossover Cable, which to choose?

Straight through vs crossover cable, which one should I choose? Usually, straight through cables are primarily used for connecting unlike devices. And crossover cables are use for connecting alike devices. Use straight through Ethernet cable for the following cabling:
Switch to router
Switch to PC or
server Hub to PC or
server
Use crossover cables for the following cabling:
Switch to switch
Switch to hub
Hub to hub
Router to router
Router Ethernet port to PC NIC
PC to PC



Conclusion on Straight Through vs Crossover Cable

Straight through and crossover cables are wired differently from each other. One easy way to tell what you have is to look at the order of the colored wires inside the RJ45 connector. If the order of the wires is the same on both ends, then you have a straight through cable. If not, then it's most likely a crossover cable or was wired wrong. At present, the straight through cable is much more popular than crossover cable and is widely used by people.
FS.COM provides a full range straight through Cat5e, Cat6, Cat6a and Cat7 Ethernet cables with many lengths and colors options.

Network cable Crimping and Testing Tools

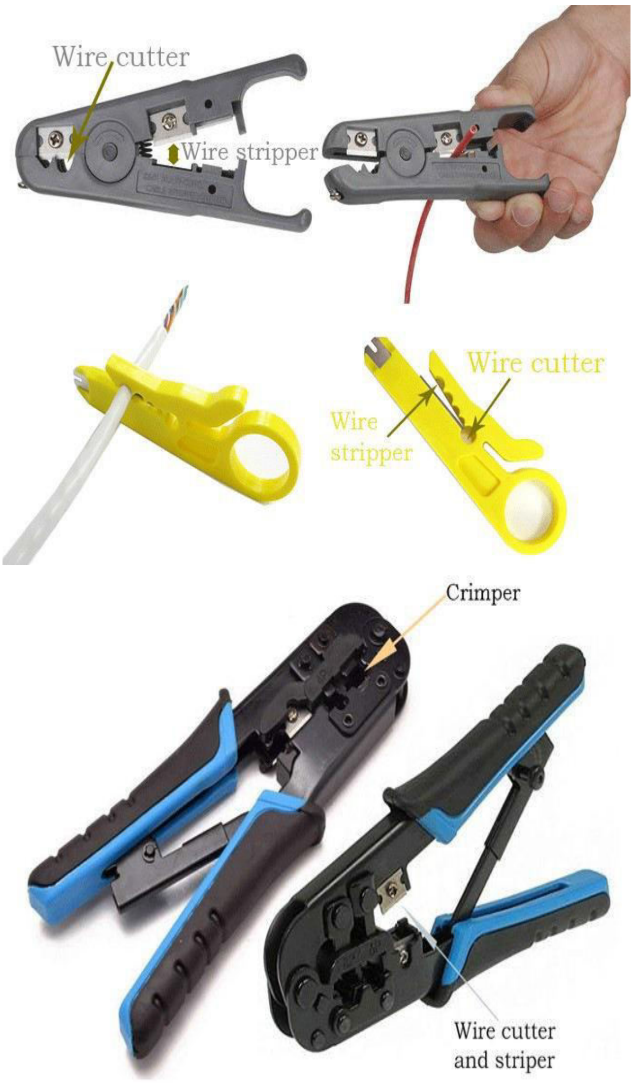
In This part of experiment explains the most common twisted-pair network cable testing and crimping tools in detail. Learn the tools that you can use to crimp and test twisted-pair network cables. Cables are the backbone of a wired network. The stability, reliability, and performance of a wired network depend on cables. Installing and maintaining cables in a wired network is a difficult task. To make this task easier, a variety of network cable crimping and testing tools are available. In this tutorial, we will not only discuss some of the most common network cable crimping and testing tools but also understand their features and functions.

Twisted-pair (STP and UTP) network cable crimping tools

Crimping tools are used for the following purposes.
1. To cut the network cable of the required length from the bundle.

2. To remove the outer and inner jackets of the network cable.
3. To attach the connectors on both ends of the cable.
Some crimping tools provide all the functionality while others provide one or two functionalities. The most common twisted-pair network cable crimping tools are described below.
Wire Cutter: - To cut the network cable of the required length from the bundle, you can use any standard wire cutter tool or can use a wire cutter tool that is specially designed for the twisted-pair cable. A twisted-pair wire cutter usually includes additional blades for stripping the wire.
Wire Stripper: - This tool is used to remove the outer and inner jackets of the network cable. Typically, you do not need to purchase this tool separately as all standard twisted-pair wire cutters are equipped with wire-strippers.

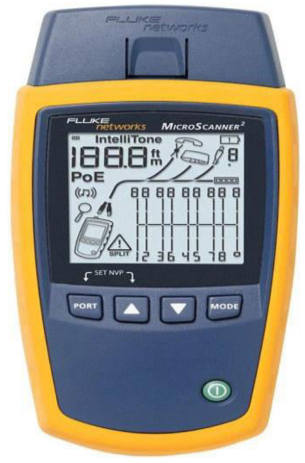
Crimp tool: - This tool is used to attach the connectors to the cable. Typically, this tool also includes a wire-cutter and wire-stripper. So if you buy a crimp tool, you don't have to buy a wire-cutter and wire-striper separately.



Which tool you should buy depends on your requirements and budget. For example, if you want to install a dozen network cables, you can buy less expensive tools such as a low-cost wire stripper and a cheap crimp device. But if you are in a network cable setting up business or have a medium or large-sized network, you should buy a crimping tool that has a built-in a wire stripper and wire cutter. A high-quality twisted-pair cable crimping tool will cost you around $100 but will save you many headaches in the long run.

Network cable testing and troubleshooting tools

A network cable testing and troubleshooting tool is used for the following purposes. To measure the length of a segment or network cable.
To detect loose connectors.
To identify an un-labeled network cable from all network cables. To find a break in the network cable.
To certify the cable installation.
The following section describes the most common network cable testing and troubleshooting tools.



Cable certifier

This device thoroughly tests a network cable and certifies that the cable installation meets a special wiring standard such as Cat 5e, Cat 6, Cat 6a, and so forth. This device can check and test total segment length, crosstalk, noise, wiremap,
resistance, impedance, and the capability to transfer data at the maximum frequency rated for the cable.

Time domain reflectometer

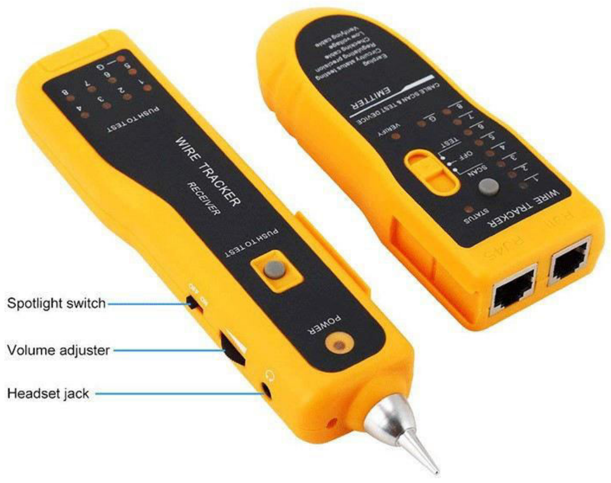
This device is used to measure the length of a network cable as well as the breaks in the cable. This device transmits a signal on one end and measures the time the signal takes to reach the end of the cable. You can also use this device to find breaks in the cable. For example, this device can tell you approximately how far the break is located in the cable.

Basic cable tester

If you can't afford a network cable certifier, you can buy and u your network cables. Besides certifying the cable installation, this device provides all remaining functionalities of a network cable certifier. It can test cable length, cross talk, and breaks in the cable. It can also check whether the connectors on both ends of a network cable are properly attached or not.

Tone generator and the probe

This device is used to trace the unlabeled network cables. This device comes in two pieces: the tone generator and the probe. The tone generator generates tones or signals and places them on the network cable. The probe detects these signals on the other end of the cable.
You can use this tool to identify network cables that run from a central location to remote locations. For example, if you are working on a patch-panel or switch and trying to figure out which network cable connects back to an end-device (such as a PC), then you can use this device. Place a tone generator at one end of the connection (end- device), and use the probe on another side (switch or patch-panel) to determine which network cable the tone generator is connected to.



Network IP:

All the computers of the world in the Internet network communicate with each other with underground or underwater cables or wirelessly. If I want to download a file from the internet or load a web page or literally do anything related to the internet, my computer must have an address so that other computers can find and locate mine in order to deliver that particular file or webpage that I am requesting. In technical terms, that address is called **IP Address or Internet Protocol Address**.

Types of IP Address

IP Address is of two types:

1. IPv4: Internet Protocol version 4. It consists of 4 numbers separated by the dots. Each number can be from 0-255 in decimal numbers. But computers do not understand decimal numbers, they instead change them to binary numbers which are only 0 and 1. Therefore, in binary, this (0-255) range can be written as (00000000 – 11111111). Since each number N can be represented by a group of 8 digit binary digits. So, a whole IPv4 binary address can be represented by 32-bits of binary digits. In IPv4, a unique sequence of bits is assigned to a computer, so a total of (2^32) devices approximately = 4,294,967,296 can be assigned with IPv4.
IPv4 can be written as:

189.123.123.90

Classes of IPv4 Address: There are around 4.3 billion IPv4 addresses and managing all those addresses without any scheme is next to impossible. Let's understand it with a simple example. If you have to find a word from a language dictionary, how long will you take? Usually, you will take less than 5 minutes to find that word. You are able to do this because words in the dictionary are organized in alphabetical order. If you have to find out the same word from a dictionary that doesn't use any sequence or order to organize the words, it will take an eternity to find the word. If a dictionary with one billion words without order can be so disastrous, then you can imagine the pain behind finding an address from 4.3 billion addresses. For easier management and assignment IP addresses are organized in numeric order and divided into the following 5 classes :

IP Class	Address Range	Maximum number of networks
Class A	0-127	128
Class B	128-191	16384
Class C	192-223	2097157
Class D	224-239	Reserve for multitasking
Class E	240-254	Reserved for Research and development

2. IPv6: But, there is a problem with the IPv4 address. With IPv4, we can connect only the above number of 4 billion devices uniquely, and apparently, there are much more devices in the world to be connected to the internet. So, gradually we are making our way to **IPv6 Address** which is a 128-bit IP address. In human-friendly form, IPv6 is written as a group of 8 hexadecimal numbers separated with colons(:). But in the computer-friendly form, it can be written as 128 bits of 0s and 1s. Since, a unique sequence of binary digits is given to computers, smartphones, and other devices to be connected to the internet. So, via IPv6 a total of (2^128) devices can be assigned with unique addresses which are actually more than enough for upcoming future generations.
IPv6 can be written as:

2011:0bd9:75c5:0000:0000:6b3e:0170:8394

Classification of IP Address

An IP address is classified into the following types:

1. Public IP Address: This address is available publicly and it is assigned by your network provider to your router, which further divides it to your devices. Public IP Addresses is of two types :

- **Dynamic IP Address:** When you connect a smartphone or computer to the internet, your Internet Service Provider provides you an IP Address from the range of available IP Addresses. Now, your device has an IP Address and you can simply connect your device to the Internet and send and receive data to and from your device. The very next time when you try to connect to the internet with the same device, your provider provides you with different IP Addresses to the same device and also from the same available range. Since IP Address keeps on changing every time when you connect to the internet, it is called Dynamic IP Address.
- **Static IP Address:** Static address never changes. They serve as a permanent internet address. These are used by DNS servers. What are DNS servers? Actually, these are computers that help you to open a website on your computer. Static IP Address provides information such as device is located in which continent, which country, which city, and which Internet Service Provider provides internet connection to that particular device. Once, we know who is the ISP, we can trace the location of the device connected to the internet. Static IP Addresses provide less security than Dynamic IP Addresses because they are easier to track.

2. Private IP Address: This is an internal address of your device which are not routed to the internet and no exchange of data can take place between a private address and the internet.

3. Shared IP addresses: Many websites use shared IP addresses where the traffic is not huge and very much controllable, they decide to rent it to other similar websites so to make it cost-friendly. Several companies and email sending servers use the same IP address (within a single mail server) to cut down the cost so that they could save for the time the server is idle.

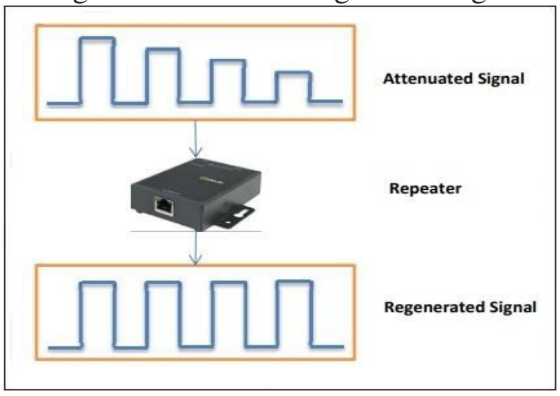
4. Dedicated IP addresses: A dedicated IP Address is an address used by a single company or an individual which gives them certain benefits using a private Secure Sockets Layer (SSL) certificate which is not in the case of a shared IP address. It allows to access the website or log in via File Transfer Protocol (FTP) by IP address instead of its domain name. It increases the performance of the website when the traffic is high. It also protects from a shared IP address that is black-listed due to spam.

TASK 2

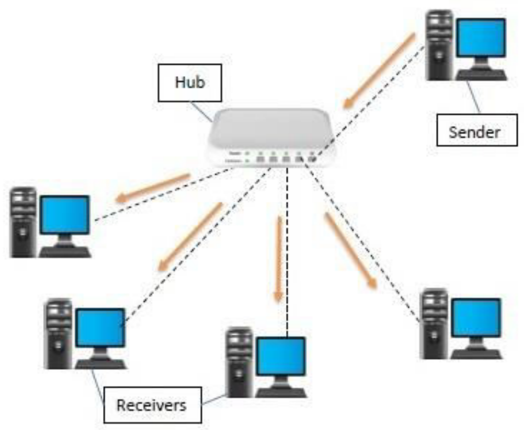
Aim: Study of following network devices in detail

a) Repeater b) Hub c)Bridge d)Router e)Gateway f)Switch

1. Repeater – A repeater operates at the physical layer. Its job is to regenerate the signal over the same network before the signal becomes too weak or corrupted so as to extend the length to which the signal can be transmitted over the same network. An important point to be noted about repeaters is that they do not amplify the signal. When the signal becomes weak, they copy the signal bit by bit and regenerate it at the original strength. It is a 2 port device.



1. **Hub** – A hub is basically a multiport repeater. A hub connects multiple wires coming from different branches, for example, the connector in star topology which connects different stations. Hubs cannot filter data, so data packets are sent to all connected devices. In other words, the collision domain of all hosts connected through Hub remains one. Also, they do not have the intelligence to find out the best path for data packets which leads to inefficiencies and wastage.



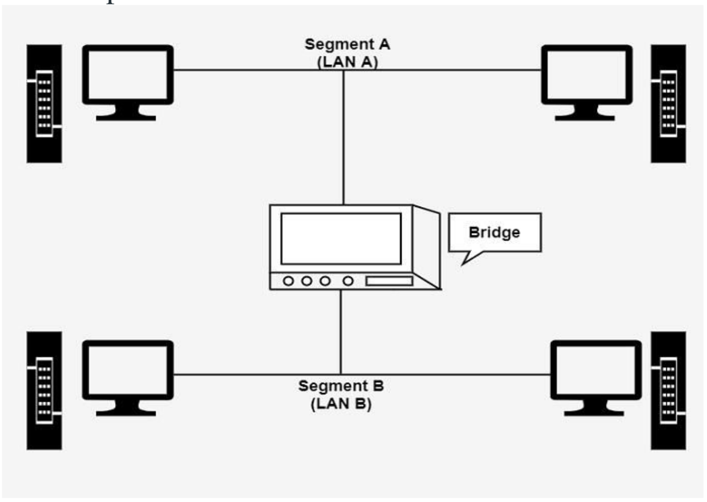
Types of Hub

- **Active Hub:-** These are the hubs that have their own power supply and can clean, boost, and relay the signal along with the network. It serves both as a repeater as well as a wiring center. These are used to extend the maximum distance between nodes.
- **Passive Hub :-** These are the hubs that collect wiring from nodes and power supply from the active hub. These hubs relay signals onto the network without cleaning and boosting them and can't be used to extend the distance between nodes.
- **Intelligent Hub :-** It works like active hubs and includes remote management capabilities. They also provide flexible data rates to network devices. It also enables an administrator to monitor the traffic passing through the hub and to configure each port in the hub.

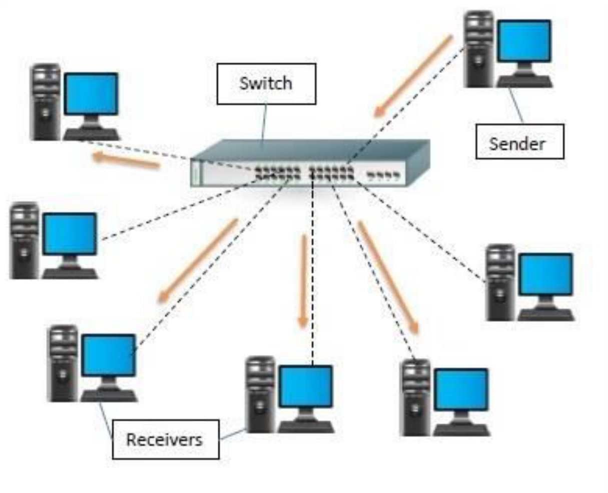
3. Bridge – A bridge operates at the data link layer. A bridge is a repeater, with add on the functionality of filtering content by reading the MAC addresses of source and destination. It is also used for interconnecting two LANs working on the same protocol. It has a single input and single output port, thus making it a 2 port device.

Types of Bridges

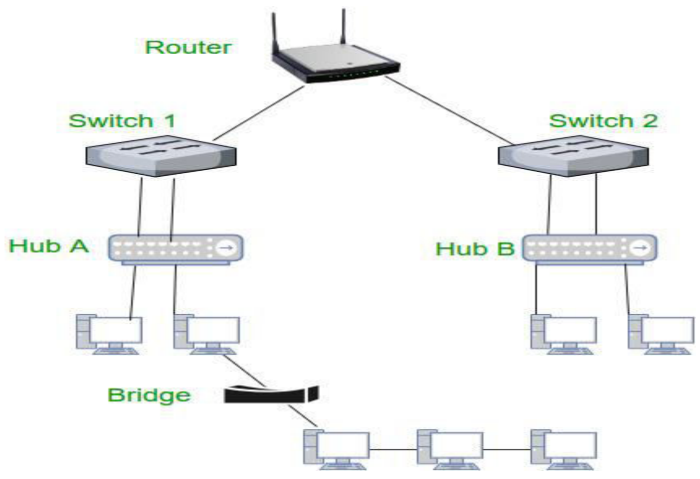
- **Transparent Bridges:-** These are the bridge in which the stations are completely unaware of the bridge's existence i.e. whether or not a bridge is added or deleted from the network, reconfiguration of the stations is unnecessary. These bridges make use of two processes i.e. bridge forwarding and bridge learning.
- **Source Routing Bridges:-** In these bridges, routing operation is performed by the source station and the frame specifies which route to follow. The host can discover the frame by sending a special frame called the discovery frame, which spreads through the entire network using all possible paths to the destination.



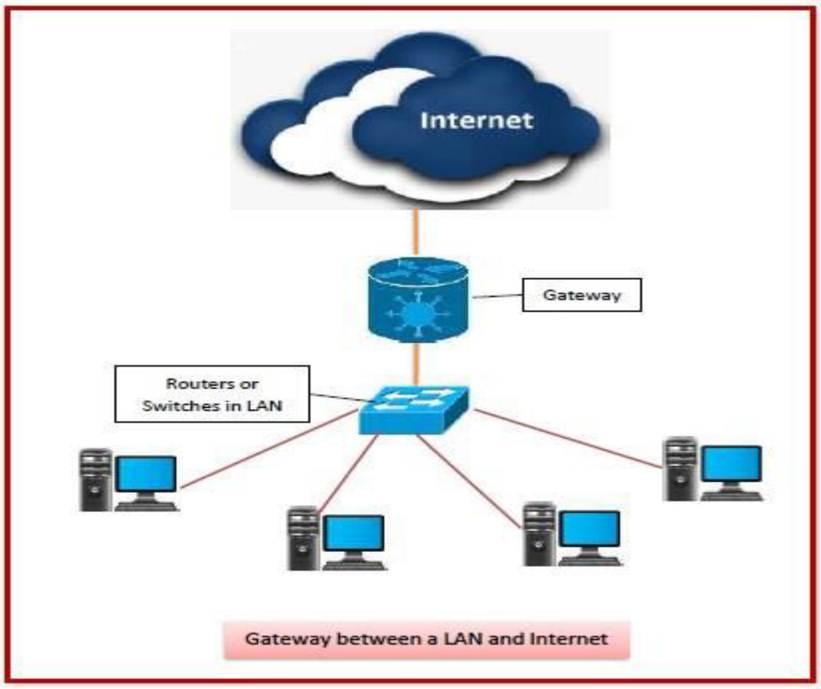
4. Switch – A switch is a multiport bridge with a buffer and a design that can boost its efficiency(a large number of ports imply less traffic) and performance. A switch is a data link layer device. The switch can perform error checking before forwarding data, which makes it very efficient as it does not forward packets that have errors and forward good packets selectively to the correct port only. In other words, the switch divides the collision domain of hosts, but broadcast domain remains the same.



5. Routers – A router is a device like a switch that routes data packets based on their IP addresses. The router is mainly a Network Layer device. Routers normally connect LANs and WANs together and have a dynamically updating routing table based on which they make decisions on routing the data packets. Router divide broadcast domains of hosts connected through it.



6. Gateway – A gateway, as the name suggests, is a passage to connect two networks together that may work upon different networking models. They basically work as the messenger agents that take data from one system, interpret it, and transfer it to another system. Gateways are also called protocol converters and can operate at any network layer. Gateways are generally more complex than switches or routers. Gateway is also called a protocol converter.


Internet
Gateway
Routers or Switches in LAN
Gateway between a LAN and Internet

TASK 3

AIM: Study and practice the basic network configuration commands

1. ifconfig Command

ifconfig (**interface configurator**) command is used to initialize an interface, assign **IP Address** to interface and **enable** or **disable** interface on demand.
With this command, you can view **IP Address** and **Hardware / MAC address** assign to interface and also **MTU** (**Maximum transmission unit**) size.

```
# ifconfig

eth0      Link encap:Ethernet  HWaddr 00:0C:29:28:FD:4C
          inet addr:192.168.50.2  Bcast:192.168.50.255
Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe28:fd4c/64
Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500
Metric:1
          RX packets:6093 errors:0 dropped:0 overruns:0
frame:0
          TX packets:4824 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6125302 (5.8 MiB)  TX bytes:536966
(524.3 KiB)
          Interrupt:18 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0
frame:0
          TX packets:8 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 txqueuelen:0
          RX bytes:480 (480.0 b)  TX bytes:480 (480.0 b)
```

ifconfig with interface (**eth0**) command only shows specific interface details like **IP Address**, **MAC Address,** etc. with `-a` option will display all available interface details if it is disabled also.

```
# ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 00:0C:29:28:FD:4C
```

```
          inet addr:192.168.50.2  Bcast:192.168.50.255
Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe28:fd4c/64
Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500
Metric:1
          RX packets:6119 errors:0 dropped:0 overruns:0
frame:0
          TX packets:4841 errors:0 dropped:0 overruns:0
carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6127464 (5.8 MiB)  TX bytes:539648
(527.0 KiB)
          Interrupt:18 Base address:0x2000
```

Set IP Address and Gateway in Linux

Assigning an **IP Address** and **Gateway** to the interface on the fly. The setting will be removed in case of a system reboot.

```
# ifconfig eth0 192.168.50.5 netmask 255.255.255.0
```

Enable or Disable Specific Interface

To **enable** or **disable** a specific Interface, we use the example command as follows.
Enable eth0

```
# ifup eth0
```

Disable eth0

```
# ifdown eth0
```

Setting MTU Size

By default **MTU** size is **1500**. We can set the required **MTU** size with the below command. Replace **XXXX** with size.

```
# ifconfig eth0 mtu XXXX
```

Set Interface in Promiscuous Mode

The network interface only received packets belonging to that particular **NIC**. If you put the interface in the **promiscuous** mode it will receive all the packets. This is very useful to capture packets and analyze them later. For this, you may require superuser access.

```
# ifconfig eth0 - promisc
```

Update: The **ifconfig** command is replaced by the IP command in most modern Linux distributions.

2. Ping Command

Ping (**Packet INternet Groper**) command is the best way to test connectivity between **two nodes**. Whether it is **Local Area Network** (**LAN**) or **Wide Area Network** (**WAN**).
Ping uses **ICMP** (**Internet Control Message Protocol**) to communicate to other devices. You can ping hostname or **ip address** using the below commands.

```
# ping 4.2.2.2

PING 4.2.2.2 (4.2.2.2) 56(84) bytes of data.
64 bytes from 4.2.2.2: icmp_seq=1 ttl=44 time=203 ms
64 bytes from 4.2.2.2: icmp_seq=2 ttl=44 time=201 ms
64 bytes from 4.2.2.2: icmp_seq=3 ttl=44 time=201 ms

OR

# ping www.tecmint.com

PING tecmint.com (50.116.66.136) 56(84) bytes of data.
64 bytes from 50.116.66.136: icmp_seq=1 ttl=47 time=284 ms
64 bytes from 50.116.66.136: icmp_seq=2 ttl=47 time=287 ms
64 bytes from 50.116.66.136: icmp_seq=3 ttl=47 time=285 ms
```

In the **Linux** ping command keep executing until you interrupt. Ping with `-c` option exit after **N** number of requests (success or error respond).

```
# ping -c 5 www.tecmint.com

PING tecmint.com (50.116.66.136) 56(84) bytes of data.
64 bytes from 50.116.66.136: icmp_seq=1 ttl=47 time=285 ms
64 bytes from 50.116.66.136: icmp_seq=2 ttl=47 time=285 ms
64 bytes from 50.116.66.136: icmp_seq=3 ttl=47 time=285 ms
64 bytes from 50.116.66.136: icmp_seq=4 ttl=47 time=285 ms
64 bytes from 50.116.66.136: icmp_seq=5 ttl=47 time=285 ms

--- tecmint.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time
4295ms
rtt min/avg/max/mdev = 285.062/285.324/285.406/0.599 ms
```

3. Traceroute Command

traceroute is a network troubleshooting utility that shows the number of hops taken to reach a destination also determines packets traveling path. Below we are tracing the route to the global **DNS server IP Address** and able to reach destination also shows the path of that packet is traveling.

```
# traceroute 4.2.2.2

traceroute to 4.2.2.2 (4.2.2.2), 30 hops max, 60 byte
packets
 1  192.168.50.1 (192.168.50.1)  0.217 ms  0.624 ms  0.133
ms
 2  227.18.106.27.mysipl.com (27.106.18.227)  2.343 ms
1.910 ms  1.799 ms
 3  221-231-119-111.mysipl.com (111.119.231.221)  4.334 ms
4.001 ms  5.619 ms
 4  10.0.0.5 (10.0.0.5)  5.386 ms  6.490 ms  6.224 ms
 5  gi0-0-0.dgw1.bom2.pacific.net.in (203.123.129.25)
7.798 ms  7.614 ms  7.378 ms
 6  115.113.165.49.static-mumbai.vsnl.net.in
(115.113.165.49)  10.852 ms  5.389 ms  4.322 ms
 7  ix-0-100.tcore1.MLV-Mumbai.as6453.net (180.87.38.5)
5.836 ms  5.590 ms  5.503 ms
 8  if-9-5.tcore1.WYN-Marseille.as6453.net (80.231.217.17)
216.909 ms  198.864 ms  201.737 ms
 9  if-2-2.tcore2.WYN-Marseille.as6453.net (80.231.217.2)
203.305 ms  203.141 ms  202.888 ms
10  if-5-2.tcore1.WV6-Madrid.as6453.net (80.231.200.6)
200.552 ms  202.463 ms  202.222 ms
11  if-8-2.tcore2.SV8-Highbridge.as6453.net (80.231.91.26)
205.446 ms  215.885 ms  202.867 ms
12  if-2-2.tcore1.SV8-Highbridge.as6453.net (80.231.139.2)
202.675 ms  201.540 ms  203.972 ms
13  if-6-2.tcore1.NJY-Newark.as6453.net (80.231.138.18)
203.732 ms  203.496 ms  202.951 ms
14  if-2-2.tcore2.NJY-Newark.as6453.net (66.198.70.2)
203.858 ms  203.373 ms  203.208 ms
15  66.198.111.26 (66.198.111.26)  201.093 ms 63.243.128.25
(63.243.128.25)  206.597 ms 66.198.111.26 (66.198.111.26)
204.178 ms
16  ae9.edge1.NewYork.Level3.net (4.68.62.185)  205.960 ms
205.740 ms  205.487 ms
17  vlan51.ebr1.NewYork2.Level3.net (4.69.138.222)  203.867
ms vlan52.ebr2.NewYork2.Level3.net (4.69.138.254)  202.850
ms vlan51.ebr1.NewYork2.Level3.net (4.69.138.222)  202.351
ms
18  ae-6-6.ebr2.NewYork1.Level3.net (4.69.141.21)  201.771
ms  201.185 ms  201.120 ms
```

```
19  ae-81-81.csw3.NewYork1.Level3.net (4.69.134.74)
202.407 ms  201.479 ms ae-92-92.csw4.NewYork1.Level3.net
(4.69.148.46)  208.145 ms
20  ae-2-70.edge2.NewYork1.Level3.net (4.69.155.80)
200.572 ms ae-4-90.edge2.NewYork1.Level3.net (4.69.155.208)
200.402 ms ae-1-60.edge2.NewYork1.Level3.net (4.69.155.16)
203.573 ms
21  b.resolvers.Level3.net (4.2.2.2)  199.725 ms  199.190
ms  202.488 ms
```

4. Netstat Command

Netstat (**Network Statistic**) command displays connection info, routing table information, etc. To display routing table information use option as `-r`.

```
# netstat -r

Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS
Window  irtt Iface
192.168.50.0    *               255.255.255.0   U         0
0          0 eth0
link-local      *               255.255.0.0     U         0
0          0 eth0
default         192.168.50.1    0.0.0.0         UG        0
0          0 eth0
```

For more examples of **Netstat Command**, please read our earlier article on 20 Netstat Command Examples in Linux.
Update: The **netstat** command is replaced by the ss (socket statistics) command in most modern Linux distributions.

5. Dig Command

Dig (**domain information groper**) query **DNS** related information like `A` Record, **CNAME**, **MX Record,** etc. This command is mainly used to troubleshoot **DNS-related** queries.

```
# dig www.tecmint.com; <<>> DiG 9.8.2rc1-RedHat-9.8.2-
0.10.rc1.el6 <<>> www.tecmint.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<
```

For more examples of **Dig Command**, please read the article on 10 Linux Dig Commands to Query DNS.

6. Nslookup Command

nslookup command is also used to find out **DNS-related** queries. The following examples show A Record (**IP Address**) of **tecmint.com**.

```
# nslookup www.tecmint.com
Server:         4.2.2.2
Address:        4.2.2.2#53

Non-authoritative answer:
www.tecmint.com canonical name = tecmint.com.
Name:   tecmint.com
Address: 50.116.66.136
```

For more **Nslookup Command**, read the article on 8 Linux Nslookup Command Examples.

7. Route Command

route command also shows and manipulates the **ip** routing table. To see the default routing table in **Linux**, type the following command.

```
# route

Kernel IP routing table
Destination     Gateway         Genmask         Flags
Metric Ref    Use Iface
192.168.50.0    *               255.255.255.0   U     0
0        0 eth0
link-local      *               255.255.0.0     U     1002
0        0 eth0
default         192.168.50.1    0.0.0.0         UG    0
0        0 eth0
```

Adding, deleting routes and default Gateway with following commands.

Add Route in Linux

```
# route add -net 10.10.10.0/24 gw 192.168.0.1
```

Delete Route in Linux

```
# route del -net 10.10.10.0/24 gw 192.168.0.1
```

Add Default Gateway in Linux

```
# route add default gw 192.168.0.1
```

8. Host Command

host command to find a name to **IP** or **IP** to name in **IPv4** or **IPv6** and also query **DNS** records.

```
# host www.google.com

www.google.com has address 173.194.38.180
www.google.com has address 173.194.38.176
www.google.com has address 173.194.38.177
www.google.com has address 173.194.38.178
www.google.com has address 173.194.38.179
www.google.com has IPv6 address 2404:6800:4003:802::1014
```

Using `-t` an option to find out DNS Resource Records like **CNAME**, **NS**, **MX**, **SOA,** etc.

```
# host -t CNAME www.redhat.com

www.redhat.com is an alias for
wildcard.redhat.com.edgekey.net.
```

9. Arp Command

ARP (Address Resolution Protocol) is useful to **view/add** the contents of the kernel's **ARP tables**. To see the default table use the command as.

```
# arp -e

Address                  HWtype  HWaddress           Flags
Mask            Iface
192.168.50.1             ether   00:50:56:c0:00:08   C
eth0
```

10. Ethtool Command

ethtool is a replacement for **mii-tool**. It is to view, setting speed and duplex of your **Network Interface Card** (**NIC**). You can set duplex permanently in **/etc/sysconfig/network-scripts/ifcfg-eth0** with **ETHTOOL_OPTS** variable.

```
# ethtool eth0

Settings for eth0:
        Current message level: 0x00000007 (7)
        Link detected: yes
```

11. Iwconfig Command

iwconfig command in **Linux** is used to configure a **wireless network interface**. You can see and set the basic **Wi-Fi** details like **SSID** channel and encryption. You can refer man page of **iwconfig** to know more.

```
# iwconfig [interface]
```

12. Hostname Command

The hostname is to identify in a network. Execute the **hostname** command to see the hostname of your box. You can set hostname permanently in **/etc/sysconfig/network**. Need to reboot box once set a proper hostname.

```
# hostname

tecmint.com
```

13. Nmcli and Nmtui Tools

The Nmcli and Nmtui tools are used to configure network settings and also used to manage network devices, create, modify, activate/deactivate, and delete network connections in Linux systems.

```
# nmcli

# nmtui
```

TASK 4

Implement on a data set of characters the three CRC polynomials

AIM: Generate CRC code for a given data

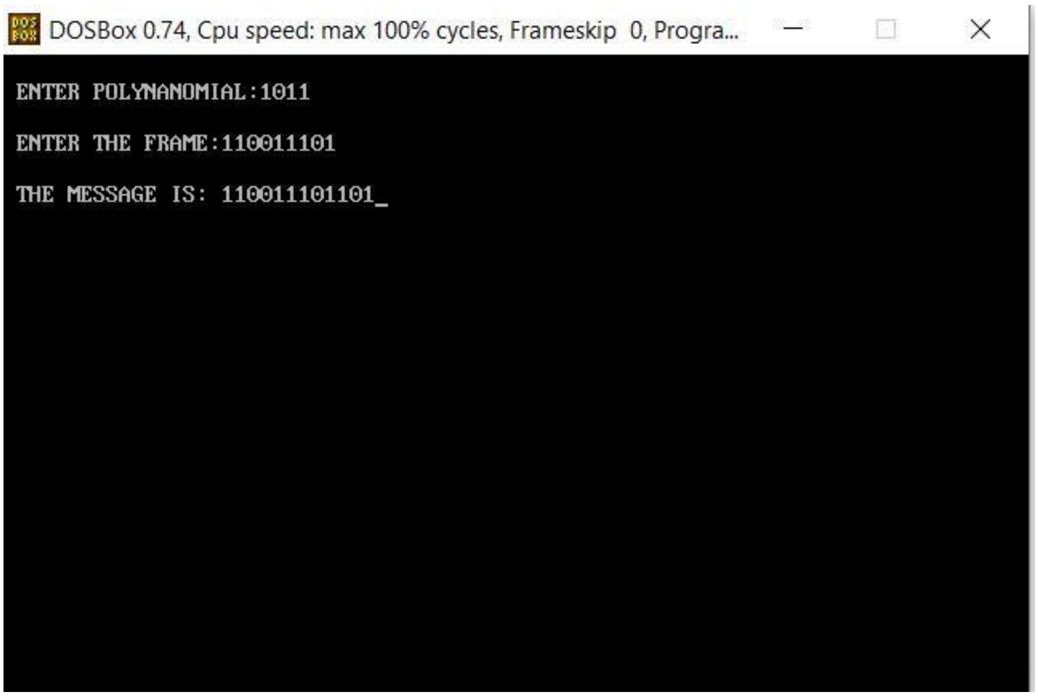
1. A string of n as is appended to the data unit. The length of predetermined divisor is n+ 1.
2. The newly formed data unit 1. A string of n as is appended to the data unit. The length of predetermined divisor is n+ 1. i.e. original data + string of n as are divided by the divisor using binary division and remainder is obtained. This remainder is called CRC.
3. Now, string of n Os appended to data unit is replaced by the CRC remainder (which is also of n bit).
4. The data unit + CRC is then transmitted to receiver.
5. The receiver on receiving it divides data unit + CRC by the same divisor & checks the remainder.
6. If the remainder of division is zero, receiver assumes that there is no error in data and it accepts it.
7. If remainder is non-zero then there is an error in data and receiver rejects it.

Program.

```
#include<stdio.h>
#include<math.h>
main()
{
int i,j,k,m,n,cl;
char a[10],b[100],c[100];
clrscr();
printf("\n ENTER POLYNANOMIAL:");
scanf("%s",a);
printf("\n ENTER THE FRAME:");
scanf("%s",b);
m=strlen(a);
n=strlen(b);
for(i=0;i<m;i++) /* To eliminat first zeros in
polynomial */
{
if(a[i]=='1')
{
m=m-i; break;
}
}
for(k=0;k<m;k++) /* To Adjust the polynomial
*/ a[k]=a[k+i];
cl=m+n-1;
for(i=0;i<n;i++) /* To copy the original frame to c[]*/
c[i]=b[i];
for(i=n;i<cl;i++) /* To add n-1 zeros at the end of frame */
```

```
 c[i]='0';
 c[i]='\0'; /*To make it as a string */
 for(i=0;i<n;i++) /* To set polynomial remainder at end of c[]*/
 if(c[i]=='1')
 {
 for(j=i,k=0;k<m;k++,j++)
 if(a[k]==c[j])
 c[j]='0';
 else c[j]='1';
 }
 for(i=0;i<n;i++) /* To copy original data in c[]*/ c[i]=b[i];
 printf("\n THE MESSAGE IS: %s",c);
 getch();
}
```

Output:

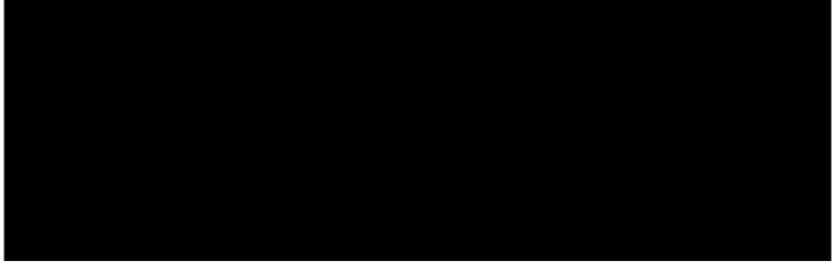


```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip  0, Progra...

ENTER POLYNANOMIAL:1011

ENTER THE FRAME:110011101

THE MESSAGE IS: 110011101101_
```

AIM: Verify the CRC code

Program:

```
 #include<stdio.h>
 #include<math.h>
 main()
 {
 int i,j,k,m,n,cl;
 char a[10],c[100]; clrscr();
 printf("\n ENTER POLYNANOMIAL:");
 scanf("%s",a);
 printf("\n ENTER THE CRC FRAME:");
 scanf("%s",c); m=strlen(a);
 cl=strlen(c);
 for(i=0;i<m;i++) /* To eliminat first zeros in polynomial
 */
 {
 if(a[i]=='1')
 { m=m-i; break; }
 }
 for(k=0;k<m;k++) /* To Adjust the polynomial */
 a[k]=a[k+i];
 n=cl-m+1;
 for(i=0;i<n;i++) /* To check polynomial remainder is
 zero or not */ if(c[i]=='1')
 {
 for(j=i,k=0;k<m;k++,j++) if(a[k]==c[j])
 c[j]='0';
 else c[j]='1';
 }
 for(i=0;i<cl;i++) /* To copy original data in c[] */
 if(c[i]=='1')
 {
 printf("\n THERE IS SOME ERROR IN MESSAGE :");
 break;
 }
 if(i==cl)
 printf("\n MESSAGE IS CORRECT");
getch();
}
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...

```
ENTER POLYNANOMIAL:01011

ENTER THE CRC FRAME:10011101011

MESSAGE IS CORRECT
```

TASK 5

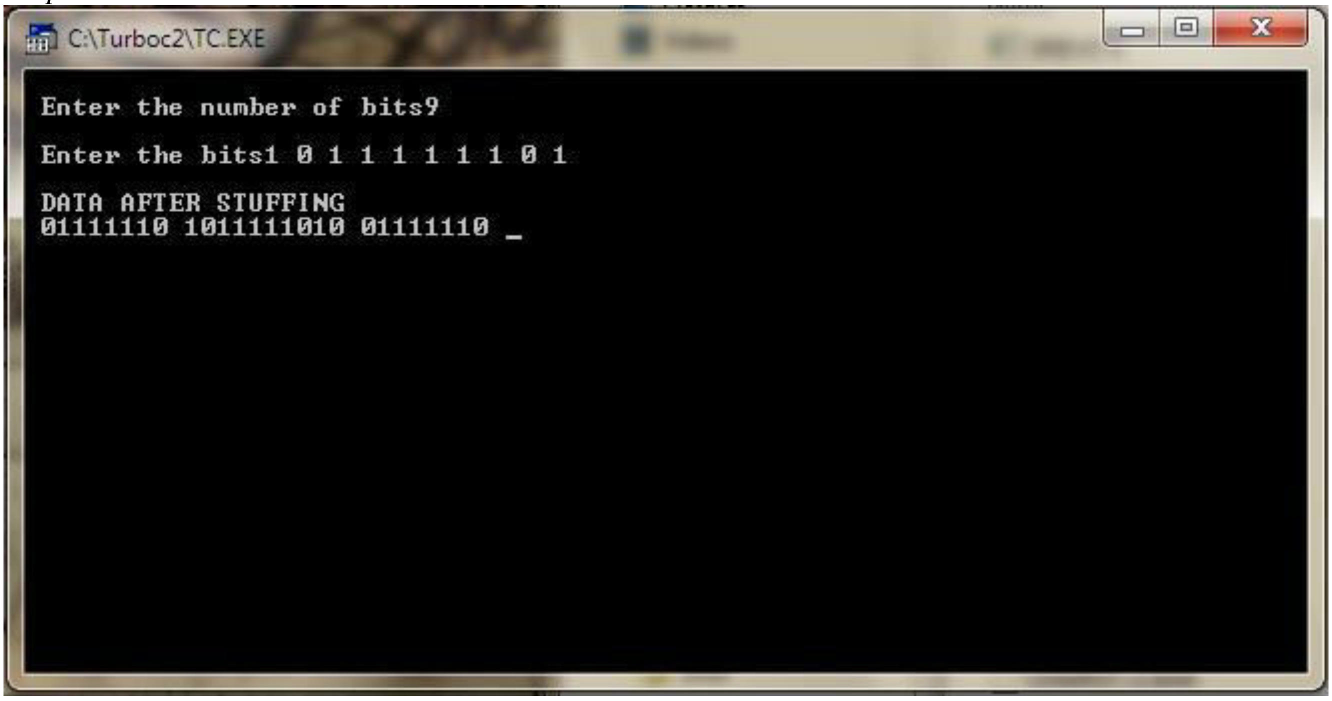
Simulate framing methods such as character stuffing and bit stuffing.

Aim: Implement the data link layer farming methods such as Bit Stuffing.

Program:

```
main()
{
int a[15];
int i,j,k,n,c=0,pos=0;
clrscr();
printf("\n Enter the number of bits");
scanf("%d",&n);
printf("\n Enter the bits");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n;i++)
{
if(a[i]==1)
{
c++;
if(c==5)
{
pos=i+1;
c=0;
for(j=n;j>=pos;j--)
{
k=j+1;
a[k]=a[j];
}
a[pos]=0;
n=n+1;
}
}
else
c=0;
}

printf("\n DATA AFTER STUFFING \n");
printf(" 01111110 ");

for(i=0;i<n;i++)
{
printf("%d",a[i]);
}
printf(" 01111110 ");
getch();
}
```

Output:



```
Enter the number of bits9

Enter the bits1 0 1 1 1 1 1 1 0 1

DATA AFTER STUFFING
01111110 1011111010 01111110 _
```

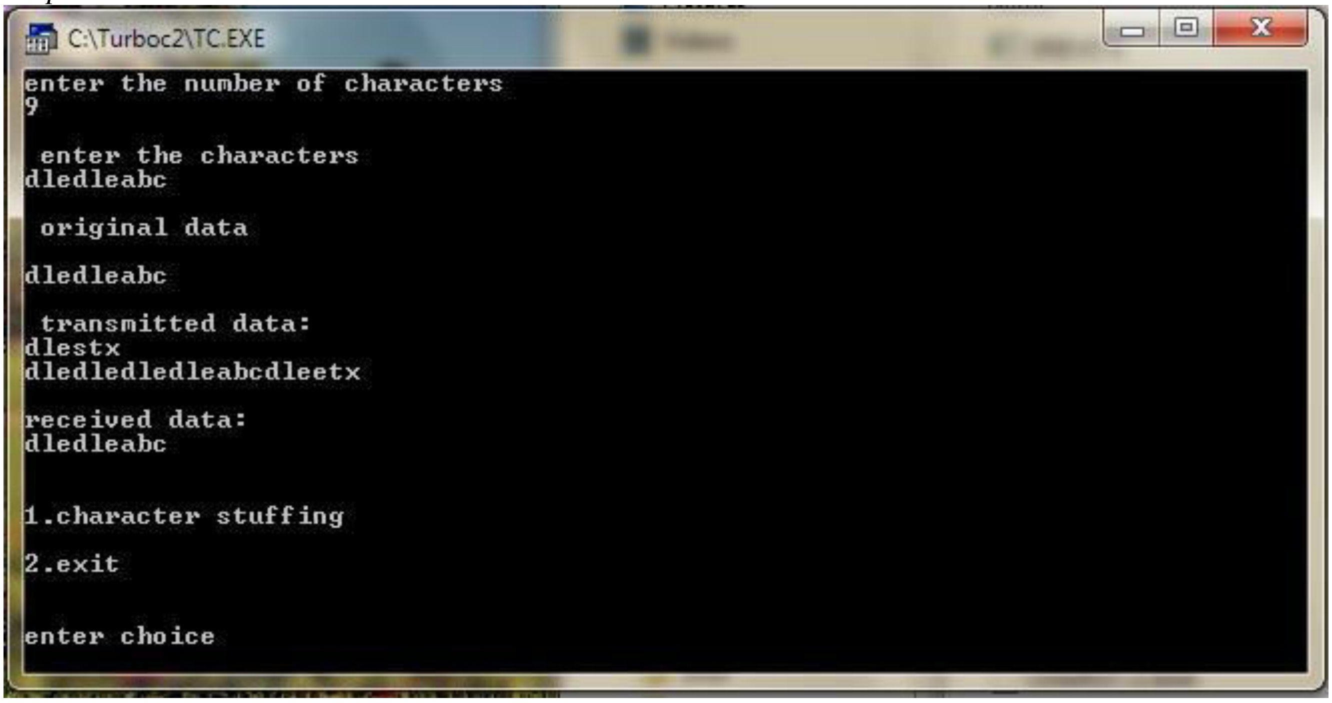
Aim: Implement the data link layer farming methods such as Character Stuffing and also De-stuff it

Program:

```
void charc(void);
void main()
{
int choice;
while(1)
{
printf("\n\n\n1.character stuffing");
printf("\n\n2.exit");
printf("\n\n\nenter choice");
scanf("%d",&choice);
printf("%d",choice);
if(choice>2)
printf("\n\n invalid option....please renter");
switch(choice)
{
case 1:
charc();
break;
case 2:
exit(0);
}
}
}
void charc(void)
{
char c[50],d[50],t[50];
int i,m,j;
clrscr();
printf("enter the number of characters\n");
scanf("%d",&m);
printf("\n enter the characters\n");
for(i=0;i<m+1;i++)
{
scanf("%c",&c[i]);
}
printf("\n original data\n");
for(i=0;i<m+1;i++)
printf("%c",c[i]);
d[0]='d';
d[1]='l';
d[2]='e';
d[3]='s';
d[4]='t';
d[5]='x';
for(i=0,j=6;i<m+1;i++,j++)
{
```

```
if((c[i]=='d'&&c[i+1]=='l'&& c[i+2]=='e'))
{
d[j]='d';
j++;
d[j]='l';
j++;
d[j]='e';
j++;
m=m+3;
}
d[j]=c[i];
}
m=m+6;
m++;
d[m]='d';
m++;
d[m]='l';
m++;
d[m]='e';
m++;
d[m]='e';
m++;
d[m]='t';
m++;
d[m]='x';
m++;
printf("\n\n transmitted data: \n");
for(i=0;i<m;i++)
{
printf("%c",d[i]);
}
for(i=6,j=0;i<m-6;i++,j++)
{
if(d[i]=='d'&&d[i+1]=='l'&&d[i+2]=='e'&&d[i+3]=='d'&&d[i+4]=='l'&&d[i+5]=='e')
i=i+3;
t[j]=d[i];
}
printf("\n\nreceived data:");
for(i=0;i<j;i++)
{printf("%c",t[i]);
}
}
```

Output:



```
enter the number of characters
9

 enter the characters
dledleabc

 original data

dledleabc

 transmitted data:
dlestx
dledledledleabcdleetx

received data:
dledleabc


1.character stuffing

2.exit


enter choice
```

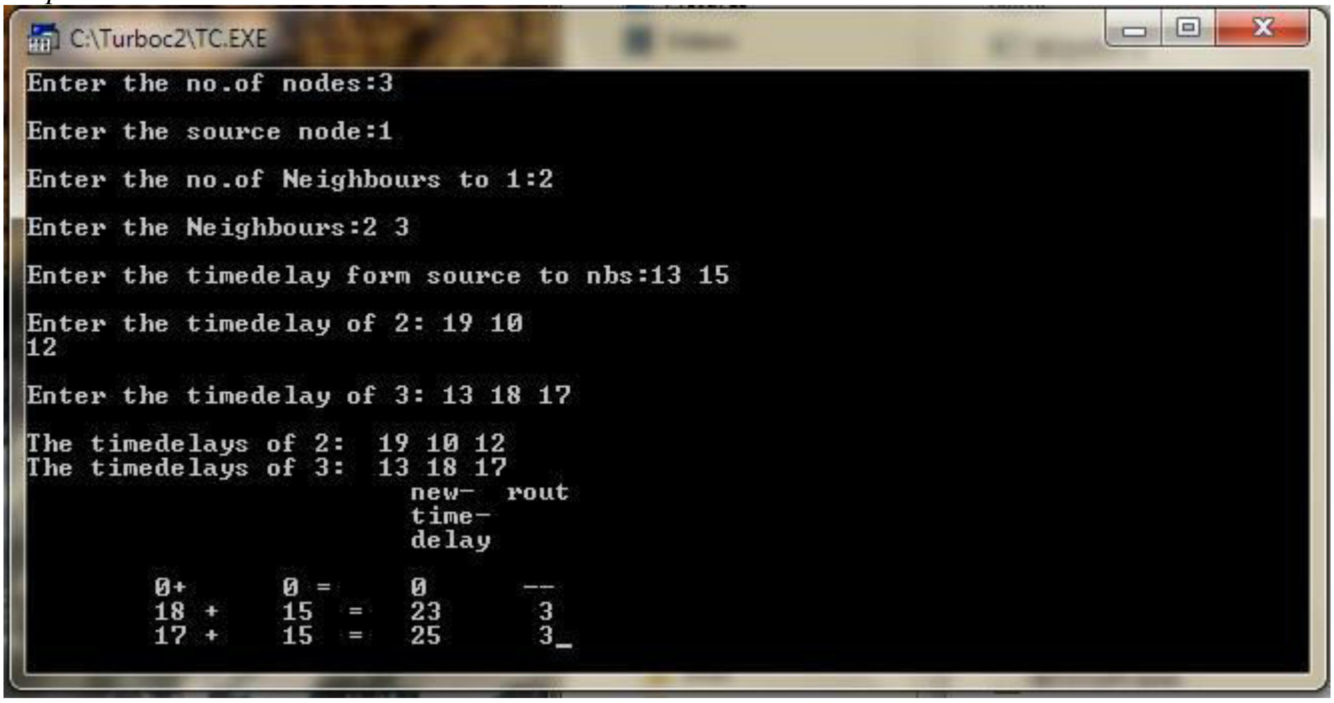
TASK 6

AIM: To Implement Dijkstra ‘s algorithm to compute the Shortest path through a graph

Program:

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

int n,s,nb,nbs[15],snbs[15],delay[15][15],i,j,temp[15],ze=0;
void min();
void main()
{
clrscr();
printf("Enter the no.of nodes:");
scanf("%d",&n);
printf("\nEnter the source node:");
scanf("%d",&s);
printf("\nEnter the no.of Neighbours to %d:",s);
scanf("%d",&nb);
printf("\nEnter the Neighbours:");
for(i=1;i<=nb;i++)
scanf("%d",&nbs[i]);
printf("\nEnter the timedelay form source to nbs:");
for(i=1;i<=nb;i++)
scanf("%d",&snbs[i]);
for(i=1;i<=nb;i++)
{
printf("\nEnter the timedelay of %d: ",nbs[i]);
for(j=1;j<=n;j++)
scanf("%d",&delay[i][j]);
}
for(i=1;i<=nb;i++)
{
printf("\nThe timedelays of %d: ",nbs[i]);
for(j=1;j<=n;j++)
printf("%3d",delay[i][j]);
}
min();
getch();
}
void min()
{
int sum,k,y=1,store=1;
printf("\n\t\t\tnew- rout");
printf("\n\t\t\ttime-");
printf("\n\t\t\tdelay");
printf("\n");
for(i=1;i<=n;i++)
{
sum=0;
k=1;
for(j=1;j<=nb;j++)
{
temp[k++]=delay[j][i];
}

sum=temp[1]+snbs[1];
for(y=2;y<=nb;y++)
{
if(sum>temp[y]+snbs[y])
{

sum=temp[y]+snbs[y];
store=y;
}
}

if(s==i)
printf("\n\t%d+\t%d =\t%d --",ze,ze,ze);
else
printf("\n\t%d +\t%d =\t%d\t%d",temp[store],snbs[store],sum,nbs[store]);
}
}
```

Output:



C:\Turboc2\TC.EXE

```
Enter the no.of nodes:3

Enter the source node:1

Enter the no.of Neighbours to 1:2

Enter the Neighbours:2 3

Enter the timedelay form source to nbs:13 15

Enter the timedelay of 2: 19 10
12

Enter the timedelay of 3: 13 18 17

The timedelays of 2:  19 10 12
The timedelays of 3:  13 18 17
                        new-  rout
                        time-
                        delay

        0+      0 =     0      --
        18 +    15  =   23      3
        17 +    15  =   25      3_
```

TASK 7

Aim: To Implement DES Encryption and Decryption

Input files to the program:

input.txt – Will contain our plain text (Max. Limit of plain text is 64kb).

key.txt – Will contain 64-bit key (Take the following key)

0001001100110100010101110111100110011011101111001101111111110001

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <time.h>

int IP[] =
{
   58, 50, 42, 34, 26, 18, 10, 2,
   60, 52, 44, 36, 28, 20, 12, 4,
   62, 54, 46, 38, 30, 22, 14, 6,
   64, 56, 48, 40, 32, 24, 16, 8,
   57, 49, 41, 33, 25, 17,  9, 1,
   59, 51, 43, 35, 27, 19, 11, 3,
   61, 53, 45, 37, 29, 21, 13, 5,
   63, 55, 47, 39, 31, 23, 15, 7
};

int E[] =
{
   32,  1,  2,  3,  4,  5,
   4,  5,  6,  7,  8,  9,
   8,  9, 10, 11, 12, 13,
   12, 13, 14, 15, 16, 17,
   16, 17, 18, 19, 20, 21,
   20, 21, 22, 23, 24, 25,
   24, 25, 26, 27, 28, 29,
   28, 29, 30, 31, 32,  1
};

int P[] =
{
   16,  7, 20, 21,
   29, 12, 28, 17,
   1, 15, 23, 26,
   5, 18, 31, 10,
   2,  8, 24, 14,
   32, 27,  3,  9,
   19, 13, 30,  6,
   22, 11,  4, 25
```

```
};

int FP[] =
{
   40, 8, 48, 16, 56, 24, 64, 32,
   39, 7, 47, 15, 55, 23, 63, 31,
   38, 6, 46, 14, 54, 22, 62, 30,
   37, 5, 45, 13, 53, 21, 61, 29,
   36, 4, 44, 12, 52, 20, 60, 28,
   35, 3, 43, 11, 51, 19, 59, 27,
   34, 2, 42, 10, 50, 18, 58, 26,
   33, 1, 41,  9, 49, 17, 57, 25
};

int S1[4][16] =
{
      14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7,
       0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
       4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
      15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13
};

int S2[4][16] =
{
   15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10,
    3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
    0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
   13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9
};

int S3[4][16] =
{
   10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8,
   13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
   13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
    1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12
};

int S4[4][16] =
{
    7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15,
   13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
   10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
    3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14
};

int S5[4][16] =
{
    2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9,
   14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
    4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
   11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3
};

int S6[4][16] =
{
   12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
   10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
   9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
   4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13
};

int S7[4][16]=
{
   4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1,
   13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
   1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
   6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12
};

int S8[4][16]=
{
   13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
   1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
   7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
   2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11
};

int PC1[] =
{
   57, 49, 41, 33, 25, 17,  9,
   1, 58, 50, 42, 34, 26, 18,
   10,  2, 59, 51, 43, 35, 27,
   19, 11,  3, 60, 52, 44, 36,
   63, 55, 47, 39, 31, 23, 15,
   7, 62, 54, 46, 38, 30, 22,
   14,  6, 61, 53, 45, 37, 29,
   21, 13,  5, 28, 20, 12,  4
};

int PC2[] =
{
   14, 17, 11, 24,  1,  5,
   3, 28, 15,  6, 21, 10,
   23, 19, 12,  4, 26,  8,
   16,  7, 27, 20, 13,  2,
   41, 52, 31, 37, 47, 55,
   30, 40, 51, 45, 33, 48,
   44, 49, 39, 56, 34, 53,
   46, 42, 50, 36, 29, 32
};
```

```
int SHIFTS[] = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };

FILE* out;
int LEFT[17][32], RIGHT[17][32];
int IPtext[64];
int EXPtext[48];
int XORtext[48];
int X[8][6];
int X2[32];
int R[32];
int key56bit[56];
int key48bit[17][48];
int CIPHER[64];
int ENCRYPTED[64];

void expansion_function(int pos, int text)
{
   for (int i = 0; i < 48; i++)
   {
      if (E[i] == pos + 1) {
         EXPtext[i] = text;
      }
   }
}

int initialPermutation(int pos, int text)
{
   int i;
   for (i = 0; i < 64; i++)
   {
      if (IP[i] == pos + 1) {
         break;
      }
   }
   IPtext[i] = text;
}

int F1(int i)
{
   int r, c, b[6];

   for (int j = 0; j < 6; j++) {
      b[j] = X[i][j];
   }

   r = b[0] * 2 + b[5];
   c = 8 * b[1] + 4 * b[2] + 2 * b[3] + b[4];

   if (i == 0) {
      return S1[r][c];
   }
```

```
    else if (i == 1) {
        return S2[r][c];
    }
    else if (i == 2) {
        return S3[r][c];
    }
    else if (i == 3) {
        return S4[r][c];
    }
    else if (i == 4) {
        return S5[r][c];
    }
    else if (i == 5) {
        return S6[r][c];
    }
    else if (i == 6) {
        return S7[r][c];
    }
    else if (i == 7) {
        return S8[r][c];
    }
}

int XOR(int a, int b) {
    return (a ^ b);
}

int ToBits(int value)
{
    int k, j, m;
    static int i;

    if (i % 32 == 0) {
        i = 0;
    }

    for (j = 3; j >= 0; j--)
    {
        m = 1 << j;
        k = value & m;
        if (k == 0) {
            X2[3 - j + i] = '0' – 48;
        }
        else {
            X2[3 - j + i] = '1' – 48;
        }
    }

    i = i + 4;
}
```

```
int SBox(int XORtext[])
{
   int k = 0;
   for (int i = 0; i < 8; i++)
   {
      for (int j = 0; j < 6; j++) {
         X[i][j] = XORtext[k++];
      }
   }

   int value;
   for (int i = 0; i < 8; i++)
   {
      value = F1(i);
      ToBits(value);
   }
}

int PBox(int pos, int text)
{
   int i;
   for (i = 0; i < 32; i++)
   {
      if (P[i] == pos + 1) {
         break;
      }
   }
   R[i] = text;
}

void cipher(int Round, int mode)
{
   for (int i = 0; i < 32; i++) {
      expansion_function(i, RIGHT[Round – 1][i]);
   }

   for (int i = 0; i < 48; i++)
   {
      if (mode == 0) {
         XORtext[i] = XOR(EXPtext[i], key48bit[Round][i]);
      }
      else {
         XORtext[i] = XOR(EXPtext[i], key48bit[17 – Round][i]);
      }
   }

   SBox(XORtext);

   for (int i = 0; i < 32; i++) {
      PBox(i, X2[i]);
   }
```

```
    for (int i = 0; i < 32; i++) {
        RIGHT[Round][i] = XOR(LEFT[Round – 1][i], R[i]);
    }
}

void finalPermutation(int pos, int text)
{
    int i;
    for (i = 0; i < 64; i++)
    {
        if (FP[i] == pos + 1) {
            break;
        }
    }
    ENCRYPTED[i] = text;
}

void convertToBinary(int n)
{
    int k, m;
    for (int i = 7; i >= 0; i--)
    {
        m = 1 << i;
        k = n & m;

        if (k == 0) {
            fprintf(out, "0");
        }
        else {
            fprintf(out, "1");
        }
    }
}

int convertCharToBit(long int n)
{
    FILE* inp = fopen("input.txt", "rb");
    out = fopen("bits.txt", "wb+");
    char ch;
    int i = n * 8;

    while (i)
    {
        ch = fgetc(inp);
        if (ch == -1) {
            break;
        }
        i--;
        convertToBinary(ch);
    }
    fclose(out);
    fclose(inp);
}

void Encryption(long int plain[])
{
    out = fopen("cipher.txt", "ab+");
    for (int i = 0; i < 64; i++) {
        initialPermutation(i, plain[i]);
    }

    for (int i = 0; i < 32; i++) {
        LEFT[0][i] = IPtext[i];
    }

    for (int i = 32; i < 64; i++) {
        RIGHT[0][i – 32] = IPtext[i];
    }

    for (int k = 1; k < 17; k++)
    {
        cipher(k, 0);

        for (int i = 0; i < 32; i++)
            LEFT[k][i] = RIGHT[k – 1][i];
    }

    for (int i = 0; i < 64; i++)
    {
        if (i < 32) {
            CIPHER[i] = RIGHT[16][i];
        }
        else {
            CIPHER[i] = LEFT[16][i – 32];
        }
        finalPermutation(i, CIPHER[i]);
    }

    for (int i = 0; i < 64; i++) {
        fprintf(out, "%d", ENCRYPTED[i]);
    }
    fclose(out);
}

void Decryption(long int plain[])
{
    out = fopen("decrypted.txt", "ab+");
    for (int i = 0; i < 64; i++) {
        initialPermutation(i, plain[i]);
    }
    for (int i = 0; i < 32; i++) {
        LEFT[0][i] = IPtext[i];
    }

    for (int i = 32; i < 64; i++) {
        RIGHT[0][i – 32] = IPtext[i];
    }

    for (int k = 1; k < 17; k++)
    {
        cipher(k, 1);

        for (int i = 0; i < 32; i++) {
            LEFT[k][i] = RIGHT[k – 1][i];
        }
    }

    for (int i = 0; i < 64; i++)
    {
        if (i < 32) {
            CIPHER[i] = RIGHT[16][i];
        } else {
            CIPHER[i] = LEFT[16][i – 32];
        }
        finalPermutation(i, CIPHER[i]);
    }

    for (int i = 0; i < 64; i++) {
        fprintf(out, "%d", ENCRYPTED[i]);
    }

    fclose(out);
}

void convertToBits(int ch[])
{
    int value = 0;
    for (int i = 7; i >= 0; i--) {
        value += (int)pow(2, i) * ch[7 – i];
    }
    fprintf(out, "%c", value);
}

int bittochar()
{
    out = fopen("result.txt", "ab+");
    for (int i = 0; i < 64; i = i + 8) {
        convertToBits(&ENCRYPTED[i]);
    }
    fclose(out);
}
```

```
void key56to48(int round, int pos, int text)
{
    int i;
    for (i = 0; i < 56; i++)
    {
        if (PC2[i] == pos + 1) {
            break;
        }
    }
    key48bit[round][i] = text;
}

int key64to56(int pos, int text)
{
    int i;
    for (i = 0; i < 56; i++)
    {
        if (PC1[i] == pos + 1) {
            break;
        }
    }
    key56bit[i] = text;
}

void key64to48(unsigned int key[])
{
    int k, backup[17][2];
    int CD[17][56];
    int C[17][28], D[17][28];

    for (int i = 0; i < 64; i++) {
        key64to56(i, key[i]);
    }

    for (int i = 0; i < 56; i++)
    {
        if (i < 28) {
            C[0][i] = key56bit[i];
        }
        else {
            D[0][i – 28] = key56bit[i];
        }
    }

    for (int x = 1; x < 17; x++)
    {
        int shift = SHIFTS[x – 1];

        for (int i = 0; i < shift; i++) {
            backup[x - 1][i] = C[x – 1][i];
        }

        for (int i = 0; i < (28 – shift); i++) {
            C[x][i] = C[x – 1][i + shift];
        }

        k = 0;
        for (int i = 28 – shift; i < 28; i++) {
            C[x][i] = backup[x – 1][k++];
        }

        for (int i = 0; i < shift; i++) {
            backup[x - 1][i] = D[x – 1][i];
        }

        for (int i = 0; i < (28 – shift); i++) {
            D[x][i] = D[x – 1][i + shift];
        }

        k = 0;
        for (int i = 28 – shift; i < 28; i++) {
            D[x][i] = backup[x – 1][k++];
        }
    }

    for (int j = 0; j < 17; j++)
    {
        for (int i = 0; i < 28; i++) {
            CD[j][i] = C[j][i];
        }

        for (int i = 28; i < 56; i++) {
            CD[j][i] = D[j][i – 28];
        }
    }

    for (int j = 1; j < 17; j++)
    {
        for (int i = 0; i < 56; i++) {
            key56to48(j, i, CD[j][i]);
        }
    }
}

void decrypt(long int n)
{
    FILE* in = fopen("cipher.txt", "rb");
    long int plain[n * 64];
    int i = -1;
    char ch;
    while (!feof(in))
    {
        ch = getc(in);
        plain[++i] = ch – 48;
    }

    for (int i = 0; i < n; i++)
    {
        Decryption(plain + i * 64);
        bittochar();
    }

    fclose(in);
}

void encrypt(long int n)
{
    FILE* in = fopen("bits.txt", "rb");

    long int plain[n * 64];
    int i = -1;
    char ch;

    while (!feof(in))
    {
        ch = getc(in);
        plain[++i] = ch – 48;
    }

    for (int i = 0; i < n; i++) {
        Encryption(plain + 64 * i);
    }

    fclose(in);
}

void create16Keys()
{
    FILE* pt = fopen("key.txt", "rb");
    unsigned int key[64];
    int i = 0, ch;

    while (!feof(pt))
    {
        ch = getc(pt);
        key[i++] = ch – 48;
    }

    key64to48(key);
    fclose(pt);
}
```

```
 long int findFileSize()
{
   FILE* inp = fopen("input.txt", "rb");
   long int size;

   if (fseek(inp, 0L, SEEK_END)) {
      perror("fseek() failed");
   }
   // size will contain number of chars in the input file.
   else {
      size = ftell(inp);
   }
   fclose(inp);

   return size;
}

int main()
{
   // destroy contents of these files (from previous runs, if any)
   out = fopen("result.txt", "wb+");
   fclose(out);

   out = fopen("decrypted.txt", "wb+");
   fclose(out);

   out = fopen("cipher.txt", "wb+");
   fclose(out);

   create16Keys();

   long int n = findFileSize() / 8;
   convertCharToBit(n);

   encrypt(n);
   decrypt(n);

   return 0;
         }
```

OUTPUT FILE:

result.txt – IT WILL CONTAIN OUR DECRYPTED TEXT.

TEMP FILES:

bits.txt – IT WILL CONTAIN OUR PLAIN TEXT CONVERTED IN BITS.

cipher.txt – IT WILL CONTAIN OUR ENCRYPTED TEXT IN BITS.

decrypted.txt – IT WILL CONTAIN OUR DECRYPTED TEXT IN BITS (SAME AS bits.txt IN CONTENT)

TASK 8

AIM: To Implement the AES Encryption and decryption

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mcrypt.h>
#include <math.h>
#include <stdint.h>
#include <stdlib.h>

int encrypt(
    void* buffer,
    int buffer_len, /* Because the plaintext could include null bytes*/
    char* IV,
    char* key,
    int key_len
){
  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 ){return 1;}

  mcrypt_generic_init(td, key, key_len, IV);
  mcrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
  mcrypt_module_close(td);

  return 0;
}

int decrypt(
    void* buffer,
    int buffer_len,
    char* IV,
    char* key,
    int key_len
){
  MCRYPT td = mcrypt_module_open("rijndael-128", NULL, "cbc", NULL);
  int blocksize = mcrypt_enc_get_block_size(td);
  if( buffer_len % blocksize != 0 ){return 1;}

  mcrypt_generic_init(td, key, key_len, IV);
  mdecrypt_generic(td, buffer, buffer_len);
  mcrypt_generic_deinit (td);
```

```
 mcrypt_module_close(td);

 return 0;
}

void display(char* ciphertext, int len){
 int v;
 for (v=0; v<len; v++){
   printf("%d ", ciphertext[v]);
 }
 printf("\n");
}

int main()
{
 MCRYPT td, td2;
 char* plaintext = "test text 123";
 char* IV = "AAAAAAAAAAAAAAAA";
 char* key = "0123456789abcdef";
 int keysize = 16; /* 128 bits */
 char* buffer;
 int buffer_len = 16;

 buffer = calloc(1, buffer_len);
 strncpy(buffer, plaintext, buffer_len);

 printf("==C==\n");
 printf("plain:   %s\n", plaintext);
 encrypt(buffer, buffer_len, IV, key, keysize);
 printf("cipher:  "); display(buffer , buffer_len);
 decrypt(buffer, buffer_len, IV, key, keysize);
 printf("decrypt: %s\n", buffer);

 return 0;
}
```

TASK 9

Aim: To Implement RSA Encryption Algorithm

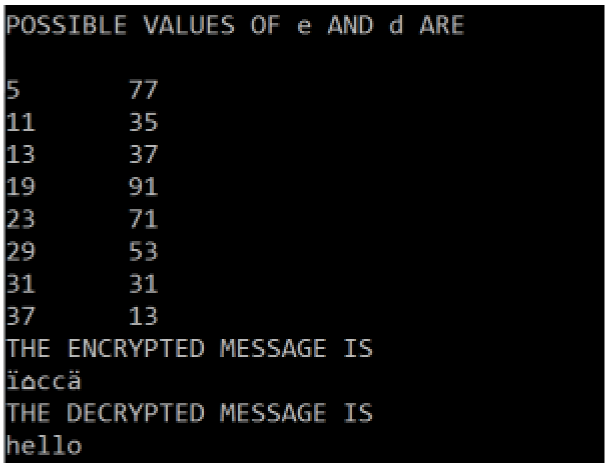
Program:

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<math.h>

#include<string.h>

long int p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;

char msg[100];

int prime(long int);

void ce();

long int cd(long int);

void encrypt();

void decrypt();

void main() {

      clrscr();

      printf("\nENTER FIRST PRIME NUMBER\n");

      scanf("%d",&p);

      flag=prime(p);

      if(flag==0) {

            printf("\nWRONG INPUT\n");

            getch();

            exit(1);

      }

      printf("\nENTER ANOTHER PRIME NUMBER\n");

      scanf("%d",&q);

      flag=prime(q);

      if(flag==0||p==q) {
            printf("\nWRONG INPUT\n");
            getch();
            exit(1);
      }
      printf("\nENTER MESSAGE\n");
      fflush(stdin);
      scanf("%s",msg);
      for (i=0;msg[i]!=NULL;i++)
      m[i]=msg[i];
      n=p*q;
      t=(p-1)*(q-1);
      ce();
      printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
      for (i=0;i<j-1;i++)
      printf("\n%ld\t%ld",e[i],d[i]);
      encrypt();
      decrypt();
      getch();
}
int prime(long int pr) {
      int i;
      j=sqrt(pr);
      for (i=2;i<=j;i++) {
            if(pr%i==0)
                return 0;
      }
      return 1;
}
void ce() {
        int k;
        k=0;
        for (i=2;i<t;i++) {
               if(t%i==0)
                   continue;
               flag=prime(i);
               if(flag==1&&i!=p&&i!=q) {
                      e[k]=i;
                      flag=cd(e[k]);
                      if(flag>0) {
                             d[k]=flag;
                             k++;
                      }
                      if(k==99)
                               break;
               }
        }
}
long int cd(long int x) {
        long int k=1;
        while(1) {
               k=k+t;
               if(k%x==0)
                   return(k/x);
        }
}
void encrypt() {
        long int pt,ct,key=e[0],k,len;
        i=0;
        len=strlen(msg);
        while(i!=len) {
               pt=m[i];
               pt=pt-96;
               k=1;
               for (j=0;j<key;j++) {
                      k=k*pt;
                      k=k%n;
               }
               temp[i]=k;
               ct=k+96;
               en[i]=ct;
               i++;
        }
        en[i]=-1;
        printf("\nTHE ENCRYPTED MESSAGE IS\n");
        for (i=0;en[i]!=-1;i++)
        printf("%c",en[i]);
}
void decrypt() {
        long int pt,ct,key=d[0],k;
        i=0;
        while(en[i]!=-1) {
               ct=temp[i];
               k=1;
               for (j=0;j<key;j++) {
                      k=k*ct;
                      k=k%n;
               }
```

```
            pt=k+96;

            m[i]=pt;

            i++;

        }

        m[i]=-1;

        printf("\nTHE DECRYPTED MESSAGE IS\n");

        for (i=0;m[i]!=-1;i++)

        printf("%c",m[i]);

}
```

Output:

```
ENTER FIRST PRIME NUMBER

7


ENTER ANOTHER PRIME NUMBER

17


ENTER MESSAGE

hello
```



```
POSSIBLE VALUES OF e AND d ARE

5       77
11      35
13      37
19      91
23      71
29      53
31      31
37      13
THE ENCRYPTED MESSAGE IS
ï⌂ccä
THE DECRYPTED MESSAGE IS
hello
```

TASK 10

Aim: To Implement Socket Programming and Client – Server model.

Client Program: tcpc.c

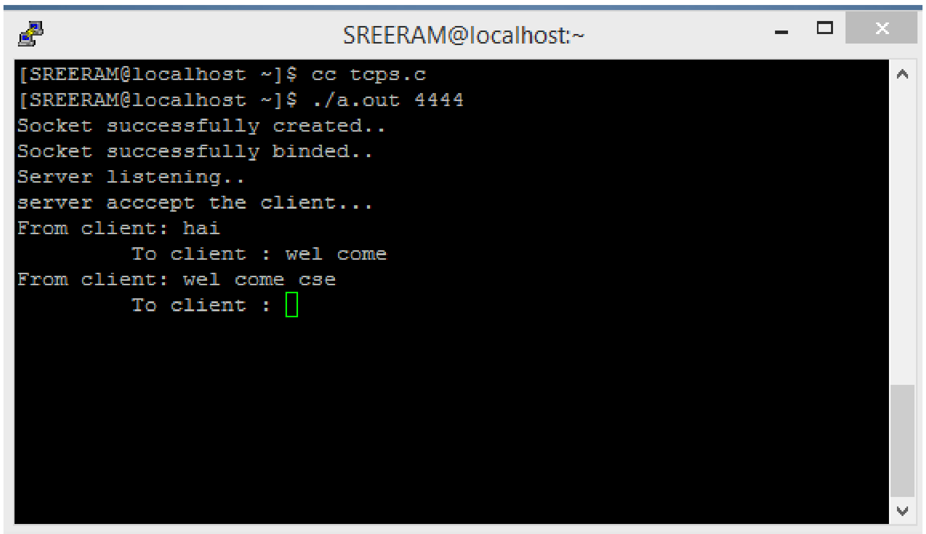
```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>

#define MAX 80
#define PORT 2222
#define SA struct isockaddr void func(int sockfd)
{
char buff[MAX]; int n;
for (;;) {
bzero(buff, sizeof(buff)); printf("Enter the string : "); n = 0;
while ((buff[n++] = getchar()) != '\n')
;
write(sockfd, buff, sizeof(buff)); bzero(buff, sizeof(buff)); read(sockfd, buff,
sizeof(buff)); printf("From Server : %s", buff);
if ((strncmp(buff, "exit", 4)) == 0) {
printf("Client Exit...\n"); break;
}
}
}

int main()
{
int sockfd, connfd;
struct sockaddr_in servaddr, cli;

// socket create and varification
sockfd = socket(AF_INET, SOCK_STREAM, 0); if (sockfd == -1) {
printf("socket creation failed...\n"); exit(0);

}
else
printf("Socket successfully created..\n");

bzero(&servaddr, sizeof(servaddr));

// assign IP, PORT servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1"); servaddr.sin_port = htons(PORT);

// connect the client socket to server socket
if(connect(sockfd,(SA*)&servaddr,sizeof(servaddr))!= 0)
{

}
else
printf("connection with the server failed...\n"); exit(0);


printf("connected to the server..\n");


// function for chat func(sockfd);

// close the socket close(sockfd);
}
```

Server Program: tcps.c

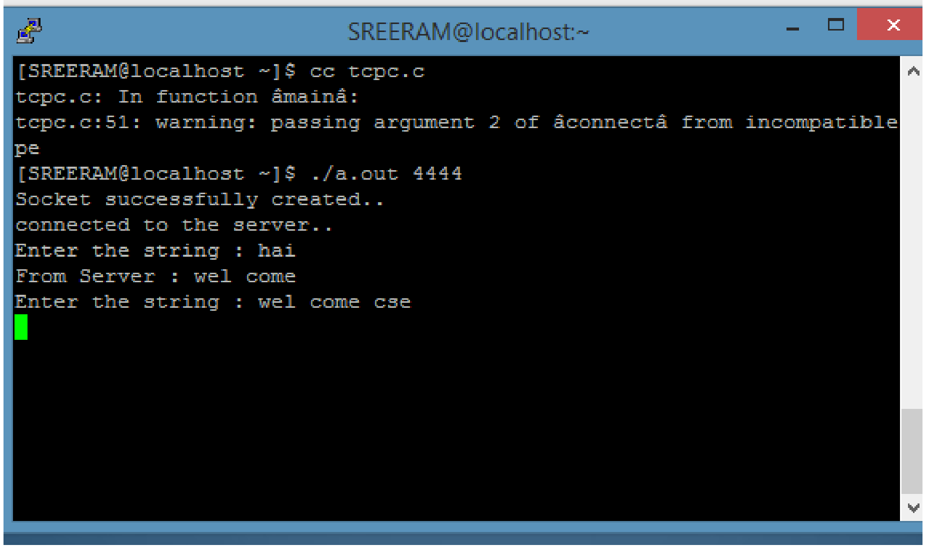
```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80

#define PORT 2222
#define SA struct sockaddr

void func(int sockfd)
{
char buff[MAX]; int n;
for (;;) { bzero(buff, MAX);

read(sockfd, buff, sizeof(buff));

printf("From client: %s\t To client : ", buff); bzero(buff, MAX);
n = 0;

while ((buff[n++] = getchar()) != '\n')
;
write(sockfd, buff, sizeof(buff)); if (strncmp("exit", buff, 4) == 0) {
printf("Server Exit...\n");
break;
}
}
}

int main()
{
int sockfd, connfd, len;
struct sockaddr_in servaddr, cli;
sockfd = socket(AF_INET, SOCK_STREAM, 0); if (sockfd == -1) {
printf("socket creation failed...\n"); exit(0);
```

```
        }
        else
        printf("Socket successfully created..\n");
        bzero(&servaddr, sizeof(servaddr));

        servaddr.sin_family = AF_INET; servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);

        if((bind(sockfd,(SA*)&servaddr, sizeof(servaddr)))!= 0)
        {
         }
        else

        printf("socket bind failed...\n"); exit(0);
        printf("Socket successfully binded..\n");
        if ((listen(sockfd, 5)) != 0) { printf("Listen failed...\n"); exit(0);

        }
        else

        printf("Server listening..\n");

        len = sizeof(cli);
        connfd = accept(sockfd, (SA*)&cli, &len); if (connfd< 0) {
        printf("server acccept failed...\n"); exit(0);

        }
        else



        printf("server acccept the client...\n");

        func(connfd);

        close(sockfd);
        }
```

OUTPUT:


SREERAM@localhost:~

```
[SREERAM@localhost ~]$ cc tcps.c
[SREERAM@localhost ~]$ ./a.out 4444
Socket successfully created..
Socket successfully binded..
Server listening..
server acccept the client...
From client: hai
         To client : wel come
From client: wel come cse
         To client :
```


SREERAM@localhost:~

```
[SREERAM@localhost ~]$ cc tcpc.c
tcpc.c: In function âmainâ:
tcpc.c:51: warning: passing argument 2 of âconnectâ from incompatible
pe
[SREERAM@localhost ~]$ ./a.out 4444
Socket successfully created..
connected to the server..
Enter the string : hai
From Server : wel come
Enter the string : wel come cse
```

TASK 11

Aim: To implement a socket program (using c) for interaction between server and client processes using Unix Domain sockets.

CLIENT:3udpc.c

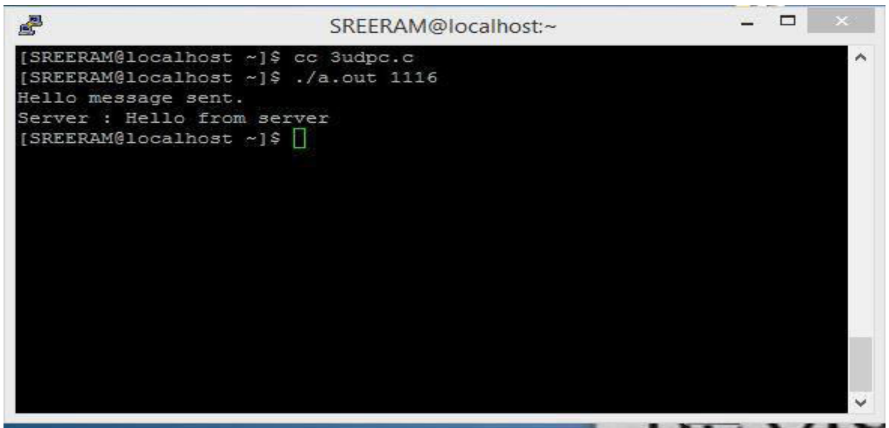
```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT 8985

#define MAXLINE 1024

// Driver code int main() {

int sockfd;

char buffer[MAXLINE];

char *hello = "Hello from client"; struct sockaddr_in           servaddr;

// Creating socket file descriptor

if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) { perror("socket
creation failed"); exit(EXIT_FAILURE);

}

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

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

int n, len;

sendto(sockfd, (const char *)hello, strlen(hello), MSG_CONFIRM, (const
struct sockaddr *) &servaddr, sizeof(servaddr));

printf("Hello message sent.\n");

n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct
sockaddr *) &servaddr, &len);
buffer[n] = '\0';
printf("Server : %s\n", buffer);
close(sockfd); return 0;
}
```

SEREVER: 3udps.c

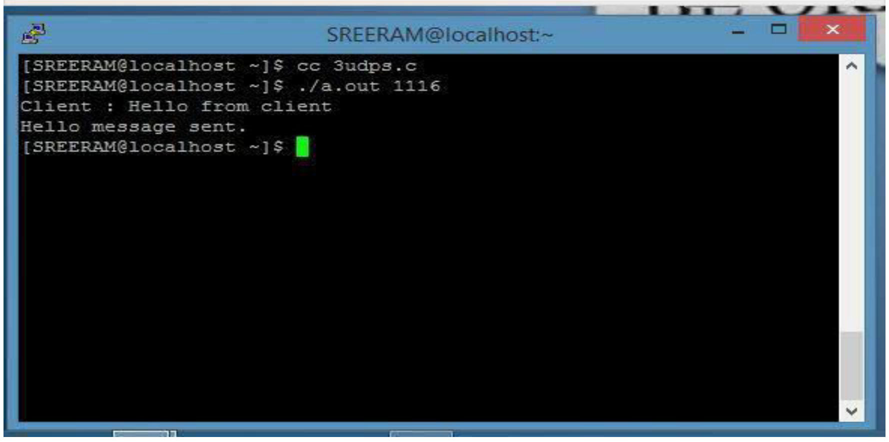
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8985
#define MAXLINE 1024
// Driver code int main() {
int sockfd;
char buffer[MAXLINE];
char *hello = "Hello from server"; struct sockaddr_in servaddr, cliaddr;
// Creating socket file descriptor
if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) { perror("socket
creation failed"); exit(EXIT_FAILURE);
}
memset(&servaddr, 0, sizeof(servaddr));
memset(&cliaddr, 0, sizeof(cliaddr));
// Filling server information servaddr.sin_family = AF_INET; // IPv4
servaddr.sin_addr.s_addr = INADDR_ANY; servaddr.sin_port =
htons(PORT);
```

```
// Bind the socket with the server address
if ( bind(sockfd, (const struct sockaddr*)&servaddr,sizeof(servaddr))<0)
{
perror("bind failed"); exit(EXIT_FAILURE);
 }
int len, n;
len = sizeof(cliaddr); //len is value/resuslt
n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, ( struct sockaddr *) &cliaddr, &len);
buffer[n] = '\0';
printf("Client : %s\n", buffer);
sendto(sockfd, (const char *)hello, strlen(hello), MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
len);
printf("Hello message sent.\n");
return 0;
```

OUTPUT:


SREERAM@localhost:~

```
[SREERAM@localhost ~]$ cc 3udpc.c
[SREERAM@localhost ~]$ ./a.out 1116
Hello message sent.
Server : Hello from server
[SREERAM@localhost ~]$ 
```


SREERAM@localhost:~

```
[SREERAM@localhost ~]$ cc 3udps.c
[SREERAM@localhost ~]$ ./a.out 1116
Client : Hello from client
Hello message sent.
[SREERAM@localhost ~]$ 
```

TASK 12

Aim: To implement a socket program (using c) for interaction between server and client processes using Internet Domain sockets.

Server program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
void error(char *msg)
{
   perror(msg);
   exit(1);
}
int main(int argc, char *argv[])
{
    int sockfd, newsockfd, portno, clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;
   if (argc < 2)
   {
      fprintf(stderr,"ERROR, no port provided\n");
      exit(1);
    }
      sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd < 0)
      error("ERROR opening socket");
      bzero((char *) &serv_addr, sizeof(serv_addr));
      portno = atoi(argv[1]);
      serv_addr.sin_family = AF_INET;
      serv_addr.sin_addr.s_addr = INADDR_ANY;
      serv_addr.sin_port = htons(portno);
   if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
      error("ERROR on binding");
      listen(sockfd,5);
      clilen = sizeof(cli_addr);
      newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    if (newsockfd < 0)
      error("ERROR on accept");
      bzero(buffer,256);
      n = read(newsockfd,buffer,255);
    if (n < 0)
      error("ERROR reading from socket");
      printf("Here is the message: %s\n",buffer);
      n = write(newsockfd,"I got your message",18);
    if (n < 0) error("ERROR writing to socket");
      return 0;
}
```

Client Program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
void error(char *msg)
{    perror(msg);
    exit(0);
}
int main(int argc, char *argv[])
{
  int sockfd, portno, n;
  struct sockaddr_in serv_addr;
  struct hostent *server;
  char buffer[256];
  if (argc < 3)
  {
     fprintf(stderr,"usage %s hostname port\n", argv[0]);
     exit(0);
  }
     portno = atoi(argv[2]);
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0)
     error("ERROR opening socket");
     server = gethostbyname(argv[1]);
  if (server == NULL)
  {
     fprintf(stderr,"ERROR, no such host\n");
     exit(0);
  }
     bzero((char *) &serv_addr, sizeof(serv_addr));
     serv_addr.sin_family = AF_INET;
     bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.s_addr,server->h_length);
     serv_addr.sin_port = htons(portno);
  if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
     error("ERROR connecting");
     printf("Please enter the message: ");
     bzero(buffer,256);
     fgets(buffer,255,stdin);
     n = write(sockfd,buffer,strlen(buffer));
  if (n < 0)
     error("ERROR writing to socket");
     bzero(buffer,256);
     n = read(sockfd,buffer,255);
  if (n < 0)
    error("ERROR reading from socket");
    printf("%s\n",buffer);
    return 0;
}
```

Output:

```
Student@ubuntu:~$gcc –o server2.out server2.c
Student@ubuntu:~$gcc –o client2.out client2.c
Please enter the message
Hello world
Hello world I got ur message
Student@ubuntu:~$ ./server2.out
Here is the message hello world
```