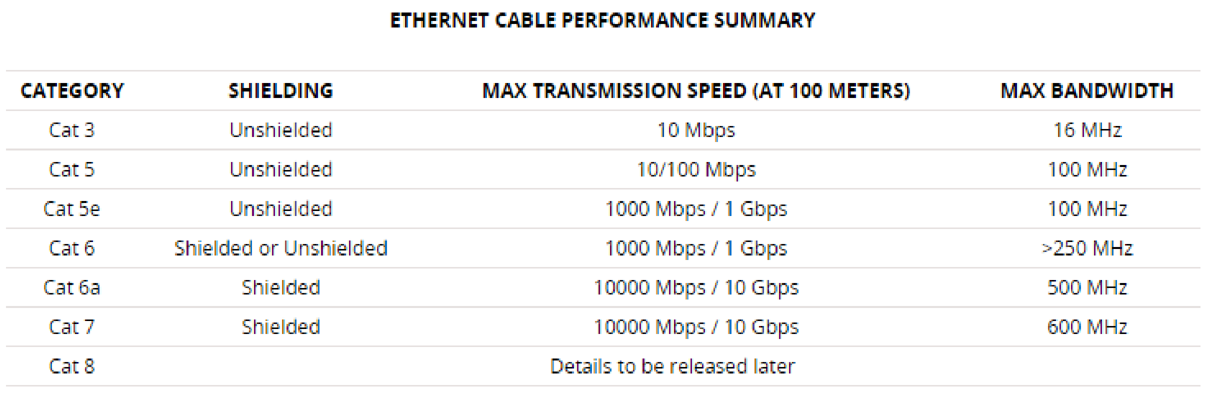
Experiment-1

1. Aim: Study of different types of network cables and practically implement cross wired cable and straight through cable using crimping tool.

Equipments Required:

1. **Cat5, Cat5e, Cat6, or Cat7 cable** - This cabling can be purchased in large spindles at stores that specialize in cabling. Cat5 cabling is the most commonly used cable used today for networks.



ETHERNET CABLE PERFORMANCE SUMMARY

CATEGORY	SHIELDING	MAX TRANSMISSION SPEED (AT 100 METERS)	MAX BANDWIDTH
Cat 3	Unshielded	10 Mbps	16 MHz
Cat 5	Unshielded	10/100 Mbps	100 MHz
Cat 5e	Unshielded	1000 Mbps / 1 Gbps	100 MHz
Cat 6	Shielded or Unshielded	1000 Mbps / 1 Gbps	>250 MHz
Cat 6a	Shielded	10000 Mbps / 10 Gbps	500 MHz
Cat 7	Shielded	10000 Mbps / 10 Gbps	600 MHz
Cat 8		Details to be released later	

2. **RJ-45 connectors** – (Registered Jack 45 connector) These connectors can be purchased at most electronic stores and computer stores and usually come in bulk packages. It is always a good idea to get more than you think you need.
3. **Crimping tool -** These tools are often purchased at electronic stores such as radio shack. To create a network cable you need a crimper that is capable of crimping a RJ-45 cable (not just a RJ-11 cable, which looks similar to a RJ-45).
4. **Wire stripper or Knife** - If you plan on making several network cables you should also consider getting a wire stripper cable of stripping Cat5, Cat6, or your cable of choice. If you do not plan on creating many network cables a knife will suffice. For simplicity and to prevent potential issues we recommend a wire stripper.

Procedure:

Step 1: Strip the cable jacket about 1.5 inch down from the end.

Step 2: Spread the four pairs of twisted wire apart. For Cat 5e, you can use the pull string to strip the jacket farther down if you need to, then cut the pull string. Cat 6 cables have a spine that will also need to be cut.

Step 3: Untwist the wire pairs and neatly align them in the T568B orientation. Be sure not to untwist them any farther down the cable than where the jacket begins; we want to leave as much of the cable twisted as possible.

Step 4: Cut the wires as straight as possible, about 0.5 inch above the end of the jacket.

Step 5: Carefully insert the wires all the way into the modular connector, making sure that each wire passes through the appropriate guides inside the connector.

Step 6: Push the connector inside the crimping tool and squeeze the crimper all the way down.

Step 7: Repeat steps 1-6 for the other end of the cable.

Step 8: To make sure you've successfully terminated each end of the cable, use a cable tester to test each pin.

The two standards for wiring Ethernet cables are T568A and T568B. T568B is the most common and is what we'll be using for our straight Ethernet cable. The tables below show the proper orientation of the colored wires to the pins.

T568A Standard

Pin 1	White/Green
Pin 2	Green
Pin 3	White/Orange
Pin 4	Blue
Pin 5	White/Blue
Pin 6	Orange
Pin 7	White/Brown
Pin 8	Brown

T568B Standard

Pin 1	White/Orange
Pin 2	Orange
Pin 3	White/Green
Pin 4	Blue
Pin 5	White/Blue
Pin 6	Green
Pin 7	White/Brown
Pin 8	Brown

Experiment-2

Aim: Program to compute number of links used on various topologies.

```
#include<stdio.h>
int main()
{
        int n;
        printf("\n Enter the number of device that need to be connected");
        scanf("%d",&n);
        printf("\n Number of links required in bus topology is %d", (n+1));
        printf("\n Number of links required in star topology is %d", (n));
        printf("\n Number of links required in mesh topology is %d", (n*(n-1))/2);
printf("\n Number of links required in star topology is %d", (n));
return 0;
}
```

Output:

```
 Enter the number of device that need to be connected4

 Number of links required in bus topology is 5
 Number of links required in star topology is 4
 Number of links required in mesh topology is 6
 Number of links required in star topology is 4
--------------------------------
Process exited after 3.004 seconds with return value 0
Press any key to continue . . .
```

Experiment-3

Aim: OSI Model encapsulation and decapsulation process by various layers

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>//for linux to run sleep command
#include<windows.h>//for windows to run sleep command

int main()
{
printf("\n\t\t ISO-OSI Model");
printf("\nLAYERS Sender Side");
printf("\nAPPLICATION LAYER !! AH !! DATA");
Sleep(1); //suspends execution of program for 1 seconds.
printf("\nPRESENTATION LAYER !! PH!!AH!!DATA");
Sleep(1);
printf("\nSESSION LAYER !! SH !!PH!!AH!!DATA");
Sleep(1);
printf("\nTRANSPORT LAYER!!TH!!SH!!PH!!AH!!DATA");
Sleep(1);
printf("\nNETWORK LAYER!!NH!!TH!!SH!!PH!!AH!!DATA");
Sleep(1);
printf("\nDATALINK LAYER!!DH!!NH!!TH!!SH!!PH!!AH!!DATA!!CRC");
Sleep(1);
printf("\nPHYSICAL LAYER 100100101010101110001010010");
Sleep(1);
printf("\n\nLAYERS Receiver Side");
printf("\nPHYSICAL LAYER 100100101010101110001010010");
Sleep(1);
printf("\nDATALINK LAYER!!DH!!NH!!TH!!SH!!PH!!AH!!DATA!!CRC");
Sleep(1);
printf("\nNETWORK LAYER!!NH!!TH!!SH!!PH!!AH!!DATA");
Sleep(1);
printf("\nTRANSPORT LAYER!!TH!!SH!!PH!!AH!!DATA");
Sleep(1);
printf("\nSESSION LAYER !! SH !!PH!!AH!!DATA");
Sleep(1);
printf("\nPRESENTATION LAYER !! PH!!AH!!DATA");
Sleep(1);
printf("\nAPPLICATION LAYER !! AH !! DATA");
Sleep(1);
return 0;
}
```

Output:

```
                ISO-OSI Model
LAYERS Sender Side
APPLICATION LAYER !! AH !! DATA
PRESENTATION LAYER !! PH!!AH!!DATA
SESSION LAYER !! SH !!PH!!AH!!DATA
TRANSPORT LAYER!!TH!!SH!!PH!!AH!!DATA
NETWORK LAYER!!NH!!TH!!SH!!PH!!AH!!DATA
DATALINK LAYER!!DH!!NH!!TH!!SH!!PH!!AH!!DATA!!CRC
PHYSICAL LAYER 100100101010101110001010010

LAYERS Receiver Side
PHYSICAL LAYER 100100101010101110001010010
DATALINK LAYER!!DH!!NH!!TH!!SH!!PH!!AH!!DATA!!CRC
NETWORK LAYER!!NH!!TH!!SH!!PH!!AH!!DATA
```

```
TRANSPORT LAYER!!TH!!SH!!PH!!AH!!DATA
SESSION LAYER !! SH !!PH!!AH!!DATA
PRESENTATION LAYER !! PH!!AH!!DATA
APPLICATION LAYER !! AH !! DATA
--------------------------------
Process exited after 0.6961 seconds with return value 0
Press any key to continue . . .
```

Experiment-4

Aim: Study of IP address, Mac Address, Port Numbers, DNS servers, Default gateway in TCP/IP configuration.

IP ADDRESS: An IP address (*internet protocol address*) is a numerical representation that uniquely identifies a specific interface on the network. Addresses in IPv4 are 32-bits long. This allows for a maximum of 4,294,967,296 (2^{32}) unique addresses. Addresses in IPv6 are 128-bits, which allows for 3.4 x 10^{38} (2^{128}) unique addresses.

The total usable address pool of both versions is reduced by various reserved addresses and other considerations.

IP addresses are binary numbers but are typically expressed in decimal form (IPv4) or hexadecimal form (IPv6) to make reading and using them easier for humans.

There are two versions of IP in use today, IPv4 and IPv6. The original IPv4 protocol is still used today on both the internet, and many corporate networks. However, the IPv4 protocol only allowed for 2^{32} addresses. This, coupled with how addresses were allocated, led to a situation where there would not be enough unique addresses for all devices connected to the internet.

IPv6 was developed by the Internet Engineering Task Force (IETF), and was formalized in 1998. This upgrade substantially increased the available address space and allowed for 2^{128} addresses. In addition, there were changes to improve the efficiency of IP packet headers, as well as improvements to routing and security.

PORT NUMBERS: In computer networking, port numbers are part of the addressing information used to identify the senders and receivers of messages. They are associated with TCP/IPnetwork connections and might be described as a sort of add-on to the IP address.

A port number is a way to identify a specific process to which an Internet or other network message is to be forwarded when it arrives at a server. For the Transmission Control Protocol and the User Datagram Protocol, a port number is a 16-bit integer that is put in the header appended to a message unit. This port number is passed logically between client and server transport layers and physically between the transport layer and the Internet Protocol layer and forwarded on.

MAC ADDRESS: Stands for "Media Access Control Address," and no, it is not related Apple Macintosh computers. A MAC address is a hardware identification number that uniquely identifies each device on a network. The MAC address is manufactured into every network card, such as an Ethernet card or Wi-Fi card, and therefore cannot be changed.
Because there are millions of networkable devices in existence, and each device needs to have a unique MAC address, there must be a very wide range of possible addresses. For this reason, MAC addresses are made up of six two-digit hexadecimal numbers, separated by colons. For example, an Ethernet card

may have a MAC address of 00:0d:83:b1:c0:8e. Fortunately, you do not need to know this address, since it is automatically recognized by most networks.

DNS SERVERS: A DNS server is a computer server that contains a database of public IP addressesand their associated hostnames, and in most cases serves to resolve, or translate, those names to IP addresses as requested. DNS servers run special software and communicate with each other using special protocols.

It's easier to remember a domain or hostname like lifewire.com than it is to remember the site's IP address numbers 151.101.129.121. When you want to navigate to the Lifewire website, all you have to type in is the URLhttps://www.lifewire.com. Conversely, computers and network devices don't work well with names when trying to locate each other on the internet. It's far more efficient and precise to use an IP address.

- Google: 8.8.8.8 & 8.8.8.44.
- Quad9: 9.9.9.9 & 149.112.112.112.
- OpenDNS: 208.67.222.222 & 208.67.220.220.
- Cloudflare: 1.1.1.1 & 1.0.0.1.
- CleanBrowsing: 185.228.168.9 & 185.228.169.9.
- Verisign: 64.6.64.6 & 64.6.65.6.

DEFAULT GATEWAY: A default gateway lets devices in one network communicate with devices in another network. If your computer, for example, is requesting an internet web page, the request first runs through your default gateway before exiting the local network to reach the internet.

An easier way to understand a default gateway might be to think of it as an intermediate device between the local network and the internet. It's necessary for transferring internal data out to the internet and then back again.

The default gateway device passes traffic from the local subnet to devices on other subnets. The default gateway often connects your local network to the internet, although internal gateways for communication within a local network also serve a useful purpose in corporate networks.

Experiment-5

Aim: Study of various network commands i.e Ping, ipconfig, tracert, getmac, nslookup.

1. **Ipconfig:** displays all current TCP/IP network configuration values.it gives you basic information to get your ip address, your's edge router ip address, DNS server ip address, DHCP server ip address Usually ur Local area connection is called Ethernet adaptor local area connection. This is the connection where u plug in ur network cable ur RJ45 cable .Default gateway is ur router address

   C:\ Ipconfig /all

2. **Ping:** allows you to send a signal to another device on the network to see if it is active.this command uses ICMP to send out an echo request to the destination device and gets back an echo response if the device u r trying to reach of in fact is active
   c:\ ping 172.16.1.1(default gateway)
   c:\ ping www.google.com
   c:\ ping www.jgfghjd.com(device that does not exist)

3. **Tracert / traceroute :** this command lets u see, the step by step route a packet takes to the destination you specify. i.e if you send a packet to google.com, before the packet actually reaches the google.com server, it will go through many different routers before it finally reaches google.com. you can also use the term hops instead of routers.
   You can say it took 10 hops to reach google.com server.

The traceroute command is used in **Linux** to map the journey that a packet of information undertakes from its source to its destination. One use for traceroute is to locate when data loss occurs throughout a network, which could signify a node that's down.

Traceroute outlines the path an IP packet follows to an internet host by launching UDP probe packets with a small TTL then listening for an ICMP "time exceeded" reply from a gateway. Start probes with a TTL of one and increase by one until you get an ICMP "port unreachable" (which means the packet arrived at its destination) or hit a max value of attempts, which defaults to 30 hops and can be changed with the **-m** flag.

When traceroute executes, it sends three probes at each TTL setting and then prints a line to the console showing the TTL, the address of the gateway, and the round-trip time of each probe. If the probe answers come from different gateways, the address of each responding system prints. If there is no response within a five-second timeout interval (changed with the **-w** flag), an asterisk is printed for that probe.

To preclude the destination host from being overwhelmed by UDP probe-packet processing, the destination port is set to a value unlikely to be used by that device. If a network or service at the destination uses that port, change the value using the **-p** flag.

Windows:

C:\ tracert www.gehu.ac.in

Linux:

Traceroute   www.gehu.ac.in

First ur edge router then ur service provider then others….

4. **Netstat :**This networking utility displays the network statistics information. netstat (network statistics) is a command line tool for monitoring network connections both incoming and outgoing as well as viewing routing tables, interface statistics etc. netstat is available on all Unix-like Operating Systems and also available on Windows OS as well. It is very useful in terms of network troubleshooting and performance measurement. netstat is one of the most basic network service debugging tools, telling you what ports are open and whether any programs are listening on ports.
   Netstat –a
   Netstat –p
   Netstat –p tcp(shows tht I am talking to somebody using TCP)
   Netstat –p udp
   Netstat –r(routing tables)

5. **Getmac:** It displays MAC addresses for the local system.
   C:\ getmac

6. **NSLOOKUP:** This networking utility query the DNS to obtain domain name or IP address mapping for any other specific DNS record.
   C:\nslookup www.google.com

Experiment-6

Aim: Implementation of parity checker program for error detection. User needs to input data and parity used (even or odd).

```
 #include<stdio.h>
#include<string.h>
int main()
{
  int i,count=0,parity;
  char data[20];
  printf("Enter sent data : ");
  scanf("%s",&data);
  printf("Choose parity\n0:Even\n1:Odd\n");
  scanf("%d",&parity);

  for(i=0;i<strlen(data);i++)
  {
     if(data[i]=='1')
       count++;
  }
  if(parity==0 && count%2==0)
    printf("No Error Detected");
  else if(parity==1 && count%2==1)
    printf("No Error Detected");
  else
    printf("Error Detected");
  return 0;
}
```

Output:

```
Enter sent data : 1000101
Choose parity
0:Even
1:Odd
1
No Error Detected
--------------------------------
Process exited after 4.282 seconds with return value 0
Press any key to continue . . .
```

Experiment - 7

Aim: Get the parity of a given number (integer).

Input: An int.

Output: it has even parity or Odd parity? If the number of 1 in that number is even, print EVEN

If the number of 1 in that number is odd, print ODD.

We have made two solutions to get the result**. (Main idea, count the no of 1 in a number)**

1. Count the number of 1 in the given number with the help of a loop until number becomes 0.

2. Loop while n is not 0 and in **loop unset one of the set bits** and invert parity.

Program 7-1:

```
#include<stdio.h>
int main()
{
   int n,count=0;
   printf("\nEnter a number to check the parity:");
   scanf("%d",&n);
   printf("\nBit representation of %d is : ",n);
   show_bits(n);
   printf("\n");
   if(n<0)  //for negative numbers.
   {
         count++;
         n= n & ~(1<<sizeof(int)*8-1);
   }
   while (n)
         {
             if(1&n)
                  count++;
             n >>= 1;
          }
   printf("No of 1 are:%d, %s",count,count%2?"ODD parity":"EVEN Parity");
   return 0;
}
void show_bits(int a)
{       int i=1;
        int size_int=sizeof(int)*8;

        printf("%s",a<0?"1":"0");

        for(i<<=size_int-2;i>0;i>>=1)
        {
           printf("%s",i&a?"1":"0");
        }
}
```

Program 7-2:

```
#include<stdio.h>
void show_bits(int);
int main()
{
    int n, parity=0;
    printf("\nEnter a number to check the parity:");
    scanf("%d",&n);
    printf("\nBit representation of %d is : ",n);
    show_bits(n);
    printf("\n");

   while (n)
          {
              parity =! parity;
              n = n & (n-1);
              }
   printf("%s",parity?"ODD parity":"EVEN Parity");
    return 0;
}

void show_bits(int a)
{
          unsigned int i=1;
          int size_int=sizeof(int)*8;
          for(i<<=size_int-1;i!=0;i>>=1)
          {
                    printf("%s",i&a?"1":"0");
          }
}
```

Experiment-8

Aim: Program to determine the class of IP address

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

// Function to find out the Class
char findClass(char str[])
{
    // storing first octet in arr[] variable
    char arr[4];
    int ip,i=0;

    while (str[i] != '.')
    {
        arr[i] = str[i];
        i++;
    }

    // converting arr[] variable into number for comparison
    ip=atoi(arr);

    if (ip >=1 && ip <= 126)
        return 'A';

    else if (ip >= 128 && ip <= 191)
        return 'B';

    else if (ip >= 192 && ip <= 223)
        return 'C';

    else if (ip >= 224 && ip <= 239)
        return 'D';

    else
        return 'E';
}
int main()
{
    char ip[15]={0};
    char ipClass ;

    printf("Enter IP Address (xxx.xxx.xxx.xxx format): ");
    scanf("%s",ip);

    ipClass = findClass(ip);
    printf("Given IP address belongs to Class %c\n", ipClass);

    return 0;
}
```

Output:

```
Enter IP Address (xxx.xxx.xxx.xxx format):

100.34.22.4

Class A Ip Address
```

Experiment-9

Aim: Program to convert IP address from dotted decimal notation to binary notation.

```
#include <stdio.h>
void DEC2BIN(int dec)
{
  for (int i = 128; i != 0; i=i>>1)
    if (dec & i)
        printf(“1”);
    else
      printf(“0”);
}

int main()
        {
        int i,j;
        int dec[4];

        printf("Enter the 4 octets of IP Address: ");
        for(i=0;i<4;i++)
           scanf(“%d”,&dec[i]);

        printf("The ip address is: %d.%d.%d.%d \n",dec[0], dec[1], dec[2],dec[3]);

        for(i=0; i<4; i++)
                {
                DEC2BIN(dec[i]);

                if(i!=3)
               printf(“.”);
                }
        }
```

Output:

```
Enter the IP Address: 1

2

3

4

The ip address is: 1.2.3.4

00000001.00000010.00000011.00000100

--------------------------------

Process exited after 7.24 seconds with return value 1

Press any key to continue . . .
```

Experiment-10

Aim: Implementation of CRC method for error control in sender side.

CRC is based on division. The actual input data is interpreted as one long binary bit stream (dividend) which is divided by another fixed binary number (divisor). **The remainder of this division is the checksum value.** The binary numbers (dividend and divisor) are not treated as normal integer values, but as binary polynomials where the actual bits are used as coefficients.

For example, the input data 0x25 = 0010 0101 is taken as $0*x^7 + 0*x^6 + 1*x^5 + 0*x^4 + 0*x^3 + 1*x^2 + 0*x^1 + 1*x^0$.

Division of polynomials differs from integer division. The underlying used arithmetic for CRC calculation is based on the **XOR (Exclusive-OR)** operation.

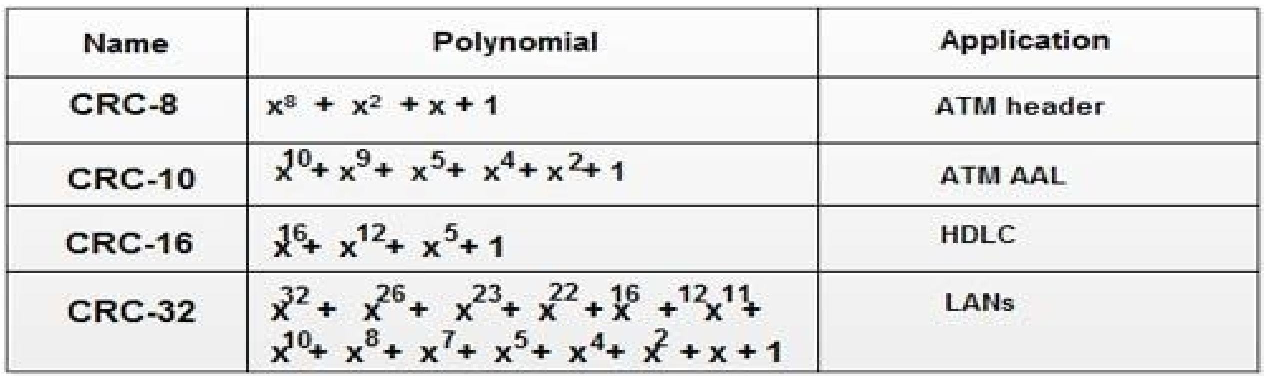
- The dividend is the complete input data (interpreted as binary stream).

- The divisor, also called *generator polynomial*, is statically defined by the used CRC algorithm.

CRC-*n* using a fixed defined generator polynomial with (n+1) bits.

- The CRC checksum value is defined as dividend % divisor

There are several different standard polynomials used by popular protocols for CRC generation. These are:



Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

- CRC can detect all single-bit errors
- CRC can detect all double-bit errors provided the divisor contains at least three logic 1's.
- CRC can detect any odd number of errors provided the divisor is a factor of x+1.
- CRC can detect all burst error of length less than the degree of the polynomial.

```
#include<stdio.h>
#include<string.h>
#define MAX 100
int main()
{
    int i,j,msg_len,gp_len;
    char text[MAX];
    char GP_key[MAX];
    char encoded[MAX]={'\0'};

    printf("Enter the Message\n");
    scanf("%s",text);
    printf("enter the generator polynomial\n");
    scanf("%s",GP_key);
    msg_len =strlen(text);
    gp_len = strlen(GP_key);
    strcpy(encoded,text);

    for(i=0;i<gp_len-1;i++)
             encoded[i+msg_len]='0';

    for(i=0;i<=msg_len;)
     {
            for(j=0;j<gp_len;j++)
          {
                    if(encoded[i+j]==GP_key[j])
                              encoded[i+j]='0';
                    else
                              encoded[i+j]='1';
          }
           while(i< (msg_len+gp_len) && encoded[i]!='1')
                    i++;
     }

    printf("The transmitted message is %s%s\n",text,encoded+msg_len);
}
```

Output:

```
enter the Message =1010

enter the generator polynomial=101

The transmitted message is 101000

--------------------------------

Process exited after 8.793 seconds with return value 0

Press any key to continue . . .
```

Experiment-11

Aim: Implementation of CRC method for error control in receiver side.

```
#include<stdio.h>
#include<string.h>
#define MAX 100
int main()
{
    int i,j,encoded_len,gp_len;
    char GP_key[MAX];
    char encoded[MAX]={'\0'};

    printf("Enter the Message\n");
    scanf("%s",encoded);
    printf("enter the generator polynomial\n");
    scanf("%s",GP_key);
    encoded_len =strlen(encoded);
    gp_len = strlen(GP_key);

    for(i=0;i< encoded_len-gp_len ;)
    {
        for(j=0;j<gp_len;j++)
             {
                    if(encoded[i+j]==GP_key[j])
                              encoded[i+j]='0';
                    else
                              encoded[i+j]='1';
             }
            while(i< encoded_len && encoded[i]!='1')
                    i++;
     }
for(i=encoded_len-gp_len;i<encoded_len;i++)
          if(encoded[i]!='0')
          {
              printf("\nERROR in message!!");
              break;
          }
 if(i==encoded_len)
   printf("\nNO ERROR in message!!");
}
```

Output:

```
Enter the Message
101000
enter the generator polynomial
101

NO ERROR in message!!
--------------------------------
Process exited after 27.71 seconds with return value 22
Press any key to continue . . .
```

Experiment-12

Aim: Program to implement character count method of Framing in data link layer.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
char s[20][10];
int x[20],i,n;
printf("Enter The No. Of Frames You Want To Send: \n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter The Frame No. %d contents\n",i);
scanf("%s",&s[i]);
}
for(i=0;i<n;i++)
{
x[i]=strlen(s[i]);
}
printf("\n Generated Code is:\n\n");
for(i=0;i<n;i++)
{
printf("%d%s",x[i],s[i]);
}
return 0;
}
```

Output:

```
Enter The No. Of Frames You Want To Send:
4

 Enter The Frame No. 0 contents
hello

 Enter The Frame No. 1 contents
how

 Enter The Frame No. 2 contents
are

 Enter The Frame No. 3 contents
graphic

 Generated Code is:

5hello3how3are7graphic
--------------------------------
Process exited after 15.39 seconds with return value 0
Press any key to continue . . .
```