Practical -1 : Implementation of Bit Stuffing & De-stuffing in Framing.

```
#include <iostream>
#include <vector>
using namespace std;

// Function for Bit Stuffing.
vector<int> bitStuffing(vector<int> &arr)
{
  // Initialize 'i' with zero to store the current index.
  int i = 0;

  // Initialize 'COUNT' with zero to store the count of consecutive 1s.
  int count = 0;

  // Declare a vector, 'ANS' to store the array after bit stuffing.
  vector<int> ans;

  // Loop through the array.
  while (i < arr.size())
  {
        // Count the number of consecutive 1s.
        while (i < arr.size() && arr[i] == 1 && count < 5)
        {
        ans.emplace_back(arr[i]);
        i++;
        count++;
        }

    // If the count of consecutive 1s is greater than five. Skip if the next element is zero, otherwise push a zero..
        if (arr[i] == 0)
        {
        ans.emplace_back(arr[i]);
        i++;
        } else if (count == 5)
    {
        ans.emplace_back(0);
        }

    count = 0;
  }
  return ans;
}

// Main.
int main()
{
  int n;
  cout << "Enter the size of the array: ";
  cin >> n;
  vector<int> arr(n);
  cout << "Enter the elements in the array: ";
  for (int i = 0; i < n; i++)
  {
        cin >> arr[i];
  }
  cout << "Before Bit Stuffing: ";
  for (int i = 0; i < n; i++)
  {
        cout << arr[i] << "  ";
  }
  cout << endl;
  arr = bitStuffing(arr);
  cout << "After Bit Stuffing: ";
  for (int i = 0; i < arr.size(); i++)
  {
        cout << arr[i] << "  ";
  }
  cout << endl;
  return 0;
}
```

Input :

```
Enter the size of the array: 12
Enter the elements in the array: 0 1 1 0 1 1 1 1 1 1 1 0
```

Output :

```
Before Bit Stuffing: 0 1 1 0 1 1 1 1 1 1 1 0
After Bit Stuffing: 0 1 1 0 1 1 1 1 1 0 1 1 0
```

Practical -2 : Implementation of RLE data compression algorithm.

```
#include<stdio.h>

#include<string.h>

void main()

{

        char str[500];

        int l,ct[100],i,j=1;


        printf("enter the string:");

        scanf("%s",str);

        l=strlen(str);

        printf("\n before RLE,string:%s\n",str);

        printf("\n After RLE,string:\n");

        for(i=0;i<l;i*=1)

        {

                j=1;

                ct[i]=1;

                for(j=i+j;str[i]==str[j];j++)

                {

                        ct[i]++;

                }

printf("\n%d%c",ct[i],str[i]);

                i=j;

        }

        printf("\n");  }
```

OUTPUT:

```
enter the string: hemlata

 before RLE,string:hemlata

 After RLE,string:

1h

1e

1m

1l

1a

1t

1a

enter the string:Archana

 before RLE,string:Archana

 After RLE,string:

1A

1r

1c

1h

1a

1n

1a
```

Practical -3 : Implementation of XOR Symmetric Cryptographic Algorithm

```
#include<stdio.h>

#include<bits/stdc++.h>

void encryptDecrypt(char inpString[])

{

        char xorKey = 'P';

        int len = strlen(inpString);

        for (int i = 0; i < len; i++)

            {

                        inpString[i] = inpString[i] ^ xorKey;

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

            }

}

 int main()

{

            char sampleString[] = "GeeksforGeeks";

            printf("Encrypted String: ");

            encryptDecrypt(sampleString);

            printf("\n");

            printf("Decrypted String: ");

            encryptDecrypt(sampleString);

            return 0;

}
```

Output

```
Encrypted String: 55;#6?"55;#
Decrypted String: GCOE,Nagaon
```

Practical -4 : Implementation of RSA Asymmetric Cryptographic Algorithm

```
#include<stdio.h>

#include<math.h>

int main()

{

        int p,q,n,phi,d,e,CT,PT,i,j;

        printf("\n enter two prime No.:");

        scanf("%d%d",&p,&q);

        n=p*q;

        phi=(p-1)*(q-1);

        printf("\n choose e such that it relativaly prime to %d",phi);

        scanf("%d",&e);

        for(d=1;d<phi;d++)

        {

                if((d*e)%phi==1)

                break;

        }

        printf("\n the private key is:{%d,%d}",d,n);

        printf("\n the public key is:{%d,%d}",e,n);

        printf("\n enter the plaintext(PT):");

        scanf("%d",&PT);

        CT=1;

        for(i=0;i<e;i++)

        CT=CT*PT%n;

        printf("\n after encryption ciphertext(CT):%d",CT);

        PT=1;

        for(i=0;i<d;i++)

        PT=PT*CT%n;

        printf("\n after decryption plaintext(PT):%d \n",PT);

        return(0);

}
```

OUTPUT:

```
 enter two prime No.:

  7

  11

 choose e such that it relativaly prime to 60

  13

 the private key is:{37,77}

 the public key is:{13,77}

 enter the plaintext(PT): 5

 after encryption ciphertext(CT):26

 after decryption plaintext(PT):5
```

Practical -5 : Implementation of TCP Socket (TCP Server and TCP Client)

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
int main()
{
 int sockfd,newsockfd,i;
 struct sockaddr_in  client_addr;
 client_addr.sin_family=AF_INET;
 char msge[20];
 client_addr.sin_port=htons(6100);

client_addr.sin_addr.s_addr=inet_addr("1
27.0.0.111");

if((sockfd=socket(AF_INET,SOCK_STR
EAM,0))<0)
 {
  printf("Client cannot open the stream
socket\n");
  exit(1);
  }
 if(connect(sockfd,(struct sockaddr
*)&client_addr, sizeof(client_addr))<0)
  {
   printf("Client cannot connect the to
server\n");
   exit(1);
  }
 printf("Enter a number : ");
 scanf("%d",&i);
 send(sockfd,&i,sizeof(i),0);
 recv(sockfd,msge,sizeof(msge),0);
 printf("Enter massage for server
%s\n",msge);
 close(sockfd);
  exit(1);
 printf("\n");
}
//server side of the socket program.
```

```
-----------------------------------------------
------------------

#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
 int main()
  {
   int             sockfd, newsockfd,i;
   struct sockaddr_in   serve_addr;
   char                    msge[20];

if((sockfd=socket(AF_INET,SOCK_STR
EAM,0))<0)
          {
          printf("Server   cannot   open   the
stream socket\n");
          exit(1);
          }
   serve_addr.sin_family=AF_INET;
   serve_addr.sin_port=htons(6100);

serve_addr.sin_addr.s_addr=htonl(INAD
DR_ANY);
                    if(bind(sockfd,       (struct
sockaddr*)&serve_addr,
sizeof(serve_addr))<0)
          {
          printf("Server cannot bind the local
address\n");
          exit(1);
          }
          listen(sockfd,5);

newsockfd=accept(sockfd,NULL,NULL)
;
          if(newsockfd<0)
          {
          printf("Server accept error.\n");
          exit(1);
          }
          recv(newsockfd,&i,sizeof(i),0);
```

```
        printf("The number : %d",i);
        if(i%2==0)
                strcpy(msge,"Even");
        else
        strcpy(msge,"Odd");

send(newsockfd,msge,sizeof(msge),0);
        close(newsockfd);
        printf("\n");
}
```

Sample Input and Output :

```
Server Started
E
n
t
e
r
M
a
s
s
a
g
e
f
o
r
S
e
r
v
e
r
G
C
O
E
N
A
G
A
O
N
R
e
p
l
y
f
r
o
m
S
e
r
v
e
r
G
C
O
E
N
A
G
A
O
N
```

Practical -6 : Implementation of UDP Socket (UDP Server and UDP Client)

Title:Socket Program For UDP client.

```
#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<string.h>

#define BUFLEN 512

#define SERVER "127.0.0.1"

int main()

{

 int sockfd,newsockfd,i;

 struct sockaddr_in  client_addr;

 int len=sizeof(client_addr);

 char buf[BUFLEN];

 char message[BUFLEN];

 char msge[20];

 memset((char *) &client_addr, 0,
sizeof(client_addr));

 client_addr.sin_family = AF_INET;


 client_addr.sin_port=htons(8888);


client_addr.sin_addr.s_addr=inet_addr("12
7.0.0.111");

 if((sockfd=socket(AF_INET,
SOCK_DGRAM, IPPROTO_UDP))<0)

  {

  printf("socket\n");

  }

while(1)

{

printf("Enter the message from server : ");

gets(message);

if (sendto(sockfd, message,
strlen(message) , 0 , (struct sockaddr *)
&client_addr, len)<0)

{

printf("sendto()");

}

memset(buf,'\0', BUFLEN);

if (recvfrom(sockfd, buf, BUFLEN, 0,
(struct sockaddr *) &client_addr, &len)<0)

{

printf("recvfrom()");

}

puts(buf);

}

close(sockfd);

printf("\n");

return 0;

}
```

Title:Socket Program For UDP server.

```
// UDP Server

#include<arpa/inet.h>

#include<stdio.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.h>

#define BUFLEN 512

 int main()

  {


   int sockfd, newsockfd;


   struct sockaddr_in
serve_addr,client_addr;

   int len=sizeof(client_addr),recv_len;

   char              msge[20];

   char buf[BUFLEN];



if((sockfd=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP))<0)

          {

          printf("\n Socket");

          }

   serve_addr.sin_family=AF_INET;

   serve_addr.sin_port=htons(8888);


serve_addr.sin_addr.s_addr=htonl(INADDR_ANY);



   if(bind(sockfd, (struct
sockaddr*)&serve_addr,
sizeof(serve_addr))<0)

          {

          printf("bind\n");

          }

          {

          printf("Waiting for data...");

          fflush(stdout);

          if ((recv_len = recvfrom(sockfd,
buf, BUFLEN, 0, (struct sockaddr *)
&client_addr, &len))<0)

          {

          printf("recvfrom()");

          }

          printf("Received packet from
%s:%d\n", inet_ntoa(client_addr.sin_addr),
ntohs(client_addr.sin_port));

          printf("Data: %s\n" , buf);



          if (sendto(sockfd, buf, recv_len, 0,
(struct sockaddr*) &client_addr, len)<0)
```

```
        {

                printf("sendto()");

        }

        }

close(sockfd);

return 0;

}
```

Sample Input and Output

```
Output:

UDP Server:

administrator@ubuntu:~$ gcc udp_server.c -o udp_server

administrator@ubuntu:~$ ./udp_server

Waiting for data...

Received packet from 127.0.0.111:54291

Data: Hello....

administrator@ubuntu:~$

UDP Client:

administrator@ubuntu:~$ gcc udp_client.c -o udp_client

/tmp/ccIF1lVj.o: In function `main':

udp_client.c:(.text+0xbd): warning: the `gets' function is dangerous and should not be used.

administrator@ubuntu:~$ ./udp_client

Enter message : Hello....
```