CO3				2										2	
CO4					3									2	
CO5						2									3

2018-19 Onwards (MR-18)	**MALLA REDDY ENGINEERING COLLEGE (Autonomous)**	**B.Tech. IV Semester**		
Code: 80519	**COMPUTER NETWORKS LAB**	**L**	**T**	**P**
Credits: 2		**-**	**1**	**2**

Prerequisite: NIL

Course Objectives:

This course provides students to understand the fundamental concepts of computer networking and communications make use of IEEE standards in the construction of LAN, build the skills of subnetting and supernetting, explain the concepts of protocols of Tranport Layer, QoS and Congestion control mechanisms and demonstrate different protocols of Application Layer.

Software Requirements: Turbo C/JDK

List of Programs:

1. Implement the data link layer farming methods:
   a) CharacterCount
   b) Character stuffing and destuffing.
   c) Bit stuffing and destuffing
2. Implement on a data set of characters the three CRC polynomials: CRC-12, CRC-16 and CRC-32.
3. Implement Parity Check using the following techniques
   a) Single Dimensional Data
   b) Multi Dimensional Data
4. Implement the Even and Odd parity.
5. Impelementation of Data Link Protocols
   a) Unrestricted Simplex Protocol
   b) Stop and wait Protocol
   c) Noisy Channel
6. Impelementation of Sliding Window Protocols
   a) One bit sliding window protocol
   b) Go Back N sliding window protocol
   c) Selective Repeat sliding window protocol
7. Write a code simulating ARP /RARP protocols
8. Impelementation of Routing Protocols
   a) Dijkstra's algorithm

b) Distance Vector routing protocol
c) Link State routing protocol

9. Implement the congestion algorithms
   a) Token bucket algorithm
   b) Leaky bucket algorithm
10. Implement DES algorithm.
11. Implement RSA algorithm.
12. Write a program to implement client-server application using TCP

TEXT BOOKS

1. Behrouz A. Forouzan, **“Data Communications and Networking”**, 4th Edition, TMH, 2006.
2. Andrew S Tanenbaum, **“Computer Networks”**, 4th Edition, Pearson Education/PHI.

REFERENCES

1. P.C .Gupta, **“Data communications and computer Networks”**, PHI.
2. S.Keshav, **“An Engineering Approach to Computer Networks”**, 2nd Edition, Pearson Education.
3. W.A. Shay, **“Understanding communications and Networks”**, 3rd Edition, Cengage Learning.

Course Outcomes:

At the end of the course, students will be able to

1. **Implement** the various protocols.
2. **Analyze** various Congestion control mechanisms.
3. **Implement** encryption mechanisms using Symmetric Key and Assymetric Key algorithms.

CO- PO Mapping
(3/2/1 indicates strength of correlation) 3-Strong, 2-Medium, 1-Weak

COs	Programme Outcomes(POs)												PSOs		
	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO10	PO11	PO12	PSO 1	PSO 2	PSO 3
CO 1			3										2		
CO 2		3												2	
CO 3				1									2		

MALLA REDDY ENGINEERING COLLEGE

(AUTONOMOUS)

Maisammaguda, Dhulapally, Secunderabad-500 014
Department of Information Technology

LAB MANUAL

`

COMPUTER NETWORKS LAB (MR18)
Code: 80519

List of Experiments:

1. Implementing the data link layer framing methods.
    - i. Character count.
2. Character Stuffing and destuffing.
3. Bit Stuffing and destuffing**.**
4. Implement on a data set of characters the three CRC polinomials:CRC-12,
    - i. CRC- 16, CRC-32.
5. Implement parity check using the following techniques.
    - i. Single dimension data.
    - ii. Multi dimension data.
6. Implement Even and Odd Parity.
7. Implementation of Data Link Protocol.
    - i. Unrestricted simplex protocol.
    - ii. Stop and wait protocol.
    - iii. Selective Repeat Sliding window protocol.
8. Implement i. Message Authentication Codes
    - ii. Cryptographic Hash Functions and Applications.
9. Implement Symmetric Key Encryption Standards (DES) and(AES).
10. Implement Diffie-Hellman Key Establishment.
11. Implement Public-Key Cryptosystems (PKCSv1.5).
12. Implement Digital Signatures.

1)

i) **character count**

AIM: To develop a c program to generate character count

Procedure :

Character-count integrity is a telecommunications term for the ability of a certain link to preserve the number of characters in a message (per unit time, in the case of a user-to-user connection). Character-count integrity is not the same as character integrity, which requires that the characters delivered be, in fact, exactly the same as they were originated.

Code :

```
#include <stdio.h>

/* count characters and input using while */
main()
{
   long nc;

   nc = 0;
   while (getchar() != EOF)
       ++nc;
   printf("%ld\n", nc);
}
```

2) CHARACTER STUFFING&DESTUFFING

Aim:

To implement the data link layer framing method character stuffing.

Problem Description:

The character stuffing method gets around the problem of re synchronization after an error by having each frame start and end with special bytes.

Character Stuffing / Byte Stuffing:

Character stuffing or byte stuffing is which an escape byte (ESC) is stuffed character stream before a flag byte in the data.

Character destuffing / Byte destuffing:

Character destuffing (or) byte destuffing is the process in which the data link layer on the receiving end removes escape byte (ESC) before the data are given to network layer.

Explanation:

To provide service to network layer, data link layer must use the services provided to it by the physical layer. The bit stream is not guaranteed to be error free.The number of bits received may be less than,equal to,or more than the number of bits transmitted,and they may have different values. It is up to the data link layer to detect and, If necessary, correct errors.

The usal approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame. When a frame arrives at the destination ,the checksum is recomputed. If the newly computed checksum is different from one contain in the frame, the data link layer knows than an error has occurred and takes steps to deal with it.

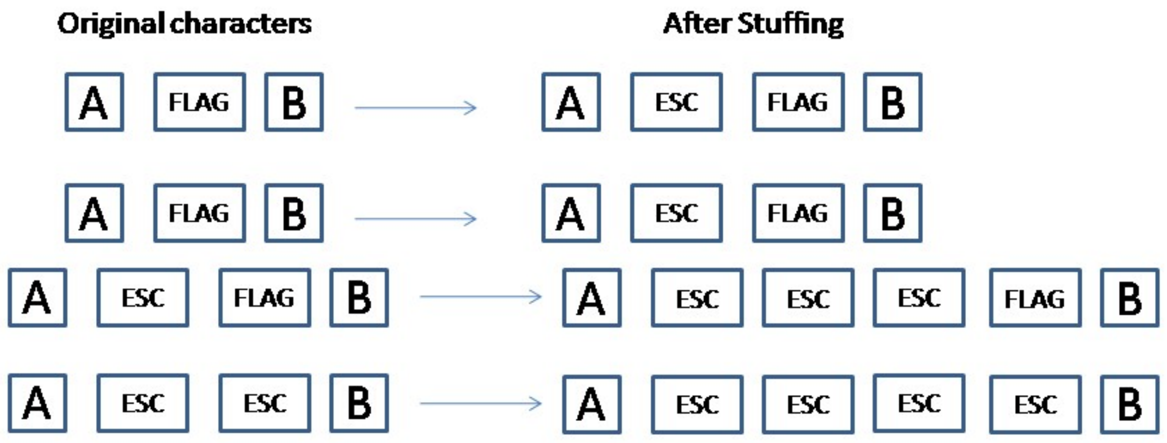
In this approach, the "flag byte" is appended at the starting and ending delimiter.

Frame Format:

FLAG	Header	Pay load field	Trailer	FLAG

Frame delimited by flag bytes

Example:



```
CharacterStuffing:
#include<stdio.h>
int pl=0;
FILE *fp,*fp1;
main()
{
void stuff();
stuff();
return;
}
void stuff()
{
int H=0,count=0,i=0,c=0,p=0,t=0;
char f[20];
char ch,prev,a[500],se[6]={'s','t','x','d','l','e'},de[6]={'e','t','x','d','l','e'};
printf("enter payload");
scanf("%d",&pl);
printf("enter the file to be stuffed");
scnaf("%c",&f);
fp=fopen("c source.txt","r");
fp1=fopen(f1,"w");
L1:while(((fscanf(fp,"%c",&ch1!=EOF)&&(w!=pl))
{
if(ch=='d')
{
a[H]=ch;
count=count+1;
}
else if(ch=='l')
{
a[H]=ch;
count=count+1;
}
else if(ch=='e')
{
a[H]=ch;
count=count+1;
if(count==3)
{
a[H+1]='d';
a[H+2]='l';
a[H+3]='e';
count=0;
if(H!=pl-3)
{
H=H+3;
count=0;
p=0;
}
else
{
prev=ch;
p=1;
}
}
}
else
{
a[H]=ch;
count=0;
}
H++;
count=0;
if(feof(fp))
c=1;
else
fseek(fp,-1,1)
L2:for(i=0;i<6;i++)
fprintf(fp,"%c",se[i]);
for(i=0;i<H;i++)
fprintf(fp1,"%c",a[i]);
for(i=0;i<6;i++)
```

```
fprintf(fp,"%c",de[i]);
fprintf(fp,"/n");
H=0;
if(p==1)
{
goto L1;
}
if(c==0)
{
p=0;
goto L1;
}
if(p==1)
{
p=0;
goto L2;
}
fcloseall();
}
```

CharacterDestuffing:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
FILE *fp,*fp1;
main()
{
void dstuff();
dstuff();
return;
}
void dstuff()
{
int i=0,count=0,j,n=0;
char ch,a[50];
size-t read;
fp=fopen("cstuff.txt","r");
fp1=fopen("out.txt","w");
while((fscanf(fp,"%c",&ch)!=EOF))
{
n++;
if(ch=='\n')
{
n=n-1;
n=n-12;
for(j=6;j<(6+n);j++)
{
fprintf(fp1,"%C",a[j]);
}
i=0;
n=0;
}
else
{
a[i]=ch;
i++;
}
}
fcloseall();
}
```

3) BIT STUFFING AND DESTUFFING

Problem: Implementing the data link layer framing methods such as the character stuffing and Bit stuffing.

Aim: To implement the data link layer framing method bit stuffing.

Problem Description: A new technique allows data frames to contain arbitrary number of bits and allows character codes with arbitrary number of bits per character.

Bit Stuffing: Bit stuffing is which an zero bit is stuffed after five consecutive ones in the input bit stream.

Bit destuffing: Bit destuffing is the process of removing the stuffed bit in the output stream.

Explanation:

To provide service to network layer, the data link layer, must use the services provided to it by the physical layer. The bit stream is not guaranteed to be error free. The number of bits received may be less than, equal to, or more than data link layer to detect and, if necessary, correct errors.

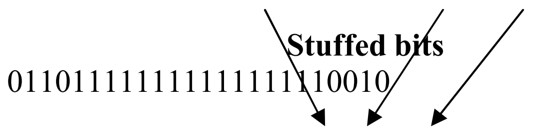
The usual approach is for the data link layer to break the bit stream up into discrete frames and compute the checksum for each frame. When a frame arrives at the destination, the checksum is re computed. If the newly computed checksum is different from one contained in the frame, the data link layer knows than an error has occurred and takes steps to deal it.

Each frame begins and ends with a special bit pattern, 01111110.When ever the sender's data link layer encounter five consecutive 1's in the data, it automatically stuffs a 0 bit in to outgoing bit stream. This bit stuffing is analogous to byte stuffing. When ever the receiver sees five consecutive incoming ones, followed by a 0 bit, it automatically dyestuffs the 0 bit.

Example:

01101111111111111110010

0101111011111011111010010



01101111111111111110010

The Original Data Bit Stuffing
The Data as they appear on the line
The data as they are stored in the receiver’s memory after destuffing.

Conclusion:

With the bit stuffing, the boundary between two frames can be unambiguously recognized by the bit pattern. Thus the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within data.

```
#include<stdio.h>
char a[100],se[8]={'0','1','1','1','1','1','1','0'};
void stuff();
void tobinary();
FILE *fp1,*fp2;
main()
{
tobinary();
stuff();
return;
}
void tobinary()
{
int v[10],i=0,a;
char ch;
fp1=fopen("soruce.txt","r");
fp2=fopen("binary.txt","w");
while(fscanf(fp1,"c",&ch)!=EOF)
{
FOR(I=0;i<7;i++)
v[i]=0;
a=ch;
i=0;
while(a!=0)
{
v[i]=a%2;
a=a%2;
i++;
}
i=6;
while(i>0)
{
fprintf(fp2,"%d",v[i]);
i--;
}
}
fcloseall();
}
void stuff()
{
int pl,x=0,count=0,i=0,p=0,c=0;
char ch,prev;
printf("enter payload");
scanf("%d",&pl);
fp1=fopen("binary.txt","r");
fp2=fopen("deatination.txt","W");
l1:while((fscanf(fp1,"%c",&ch)!=EOF)&&(x!=pl))
{
if(ch=='0')
{
a[x]=ch;
count=0;
}
elseif(ch=='1')
{
if(count=='5')
{
a[x]=0;
if(x!=pl-1)
{
x=x+1;
a[x]=ch;
count=0;
p=0;
}
else
{
prev=ch;
p=1;
}
count=count+1;
}
else
{
a[x]=ch;
count=count+1;
}
}
x++;
}
```

```
count=0;
if(feof(fp1))
c=1;
else
fseek(fp1,-1,1);
L2:for(i=0;i<8;i++)
fprintf(fp2,"%c",se[i]);
for(i=0;i<x;i++)
fprintf(fp2,"%c",a[i]);
for(i=0;i<8;i++)
fprintf(fp2,"%c",se[i]);
fprintf(fp2,"\n");
x=0;
if(p==1)
{
if(prev==0)
{
a[x]=prev;
x++;
count=0;
}
elseif(prev==1)
{
a[x]=prev;
count=count+1;
x++;
}
}
if(c==0)
{
p=0;
goto L1;
}
if(p==1)
{
p=0;
goto L2;
}
fcloseall();
}
```

BitDestuffing:

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

```
#include<string.h>
FILE *fp,*fp1;
main()
{
void dstuff();
void bintochar();
dstuff();
bintochar();
return;
}
void dstuff()
{
int i=0,count=0,pl,n=30,j;
char ch,a[50];
ssize_+ read;
printf("enter pay load");
scanf("%d",&pl);
fp=fopen("destination.txt","r");
fp=fopen("output.txt","w');
while(fgets(a,n,fp)!='\0')
{
a[strlen[a]-9]='\0';
for(i=0;a[i]!='\0';i++)
a[i]=a[i+s]
for(j=0;j<strlen(a);j++)
{
if(a[j]=='0')
{if(coumt==5)
count=0;
]
else
{
fprintf(fp1,"%',a[j]);
count=0;
}
]
if(a[j]=='1')
{
count++;
fprintf(fp1,"%c",a[j]);
}
}
}
fclose();
}
```

```
void bintochar()
{
char ch;
int a[7],p=0,j=0,i=0,c=0,sum=0;
fp=fopen("output1.txt","r");
fp=fopen("output2.txt','"w');
l1;while((fscanf(fp,"%c",&ch)!=EOF)&&(i<7))
{
if(ch=='1')
a[i]=1;
elseif(ch=='0')
a[i]=0;
i++;
}
p=6;
while(j<7)
{
if(a[j]==1)
sum=sum+pow(2,p);
j++;
p--;
}
if(feof(fp))
c=1;
if(c==0)
{
fprintf(fp,"%c",sum);
fseek(fp,-1,1);
sum=0;
j=0;
i=0;
goto l1;
}
fcloseall();
}
```

4. CRC

Problem Description:

Implement on data set of characters the there crc polynomials – crc12, crc16, crcccitt.

Aim:

To implement on data set of characters the three crc polynomials - crc12, crc16, crcccitt.

Program Description:

Error correcting codes are widely used on wireless links, which are notoriously noisy and error prone when compared to copper wire or optical fiber. The polynomial code, also know as a crc (Cycle Redundancy Check). Polynomial codes are based on treating bit strings as representations of polynomial with coefficient of 0 and 1 only.

Explanation:

A k-bit frame is regarded as the coefficient list for a polynomial with k-terms, ranging from x^{k-1} to x^0. Such a polynomial is said to be of degree k-1. The higher order bit is the coefficient of x^{k-1} .The next bit is coefficient of x^{k-2} and so on.

Polynomial arithmetic is done modulo 2,according to the rules of algebraic field theory.

Example:

```
   1 0 0 1 1 0 1 1       0 0 1 1 0 0 1 1       1 1 1 1 0 0 0 0       0 1 0 1 0 1 0 1
 + 1 1 0 0 1 0 1 0   +   1 1 0 0 1 1 0 1   -   1 0 1 0 0 1 1 0   -   1 0 1 0 1 1 1 1
 _________________      ________________      ________________      ________________
   0 1 0 1 0 0 0 1       1 1 1 1 1 1 10        0 1 0 1 0 1 1 0       1 1 1 1 1 0 1 0
 _________________      ________________      ________________      ________________
```

When polynomial is employed, the sender and receiver must agree up on a generated polynomial, g(x),in advance. Both high and low order bits of the generator must one. To compute the check sum for some frame with m bits,corresponding to the polynomial m(x),the frame must be longer than the generated polynomial.

PARITY TECHNIQUES

A parity check is the process that ensures accurate data transmission between nodes during communication. A parity bit is appended to the original data bits to create an even or odd bit number; the number of bits with value one. The source then transmits this data via a link, and bits are checked and verified at the destination. Data is considered accurate if the number of bits (even or odd) matches the number transmitted from the source.

Parity Check

Parity checking, which was created to eliminate data communication errors, is a simple method of network data verification and has an easy and understandable working mechanism.

As an example, if the original data is 1010001, there are three 1s. When even parity checking is used, a parity bit with value 1 is added to the data's left side to make the number of 1s even; transmitted data becomes 11010001. However, if odd parity checking is used, then parity bit value is zero; 01010001.

If the original data contains an even number of 1s (1101001), then parity bit of value 1 is added to the data's left side to make the number of 1s odd, if odd parity checking is used and data transmitted becomes 11101001. In case data is transmitted incorrectly, the parity bit value becomes incorrect; thus, indicating error has occurred during transmission.

Source code:

```
#include<stdio.h>
#include<string.h>
void crc();
char s[50],r[50]="";
int i,j,k,n;
```

```
main()
{
char ch;
do
{
int a=12,b=16,c=22;
printf("\t\t\t\t\n*****CRC*****\N");
printf("1.CRC12 \n 2.CRC16 \n 3.CRC CCITT\n Enter your choice:");
scnaf("%d",&n);
switch(n)
{
case 1:CRC(a);
      break;
case 2:CRC(b);
      break;
case 3:CRC(c);
      break;
default:printf("Enter correct choice \n");
break;
}
printf("\n do you want to continue:");
fflush(stdin);
scanf("%c",&ch);
}
while(ch=='y'||ch=='y');
}
void crc(int x)
{
char g[20],c;
FILE *fp,*fp1;
char fname[25],fname1[25];
printf("\n Enter file:");
scanf("%s",fname);
fp=fopen(fname,"w");
if(x==12)
strcpy(g,"1100000001111");
if(x==16)
strcpy(g,"11000000000000101");
if(x==12)
printf("\t\t \n YOU HAVE ENTERED CRC 12");
if(x==16)
printf("\t\t \n YOU HAVE ENTERED CRC 16");
if(x==22)
printf("\t\t \n YOU HAVE ENTERED CRC CCITT");
puts("\n Enter the message");
```

```
scanf("%s",s);
printf("\n entered string is: %s",s);
for(i=0;i<strlen(g)-1;i++)
strcat(s,"0");
printf("\n string after appending zero s:%S \n",s);
for(i=0;s[i]!='\0';)
{
p=strlen(r);
while((strlen(g)!=strlen(r)&&(s[i]!='\0'))
{
r[j++}==s[i++];
r[j]='\0';
}
for(j=0;strlen[g]==strlen9r00&&g[j]!='\0';j++)
{
if(g[j]==r[j])
r[j]='0';
else
r[j]='1';
}
while(r[0]=='0'&&r[0]!='\0')
{
for(j=0;r[j]!='\0';j++)
r[j]=r[j+1];
}
}
s[strlen(s)-strlen(r)]='\0';
strcat(s,r);
printf("\n string converted is :%s and stored at %s\n',s,frame);
fprintf(fp,"%s",s);
fcloseall();
printf("\n do you want to check?:");
scanf("%c",&c);
if(c=='y'||c=='y')
{
printf("\nb enter file:");
scanf("%s',frame);
fp1=fopen(frame,"r");
fgets(s,100,fp1);
strcpy(r,"\0");
for(i=0;s[i]!='0';i++)
{
j=strlen(r);
while((strlen(g)!=strlen(r)&&(s[i]!='\o'))
{
```

```
r[j++]=s[i++];
r[j]='\0';
}
for(j=0;(strlen(g)==strlen(r)&&g[i]!='\0';j++)
{

if(g[j]==r[j])
r[j]='\0';
else
r[j]='1';
}
while(r[0]='0'&& r[0]!='\0')
{
for(j=0;r[j]!='\0';i++)
r[j]=r[j+1];
}
}
s[strlen(s)-strlen(g)+1]='\0';
strcat(s,r);
fcloseall();
printf("\n transmitted string is: %s \n",s);
if(strlen(r)==0)
{
printf("\n success");
}
else
printf("\n data corrupted");
}
else
exit(0);
}
```

5) Implement parity check techniques
i) Single dimensional
AIM: To design the concept of parity check techniques
THEORY: **PARITY TECHNIQUES**

A parity check is the process that ensures accurate data transmission between nodes during communication. A parity bit is appended to the original data bits to create an even or odd bit number; the number of bits with value one. The source then transmits this data via a link, and bits are checked and verified at the destination. Data is considered accurate if the number of bits (even or odd) matches the number transmitted from the source.

Parity Check

Parity checking, which was created to eliminate data communication errors, is a simple method of network data verification and has an easy and understandable working mechanism.

As an example, if the original data is 1010001, there are three 1s. When even parity checking is used, a parity bit with value 1 is added to the data's left side to make the number of 1s even; transmitted data becomes 11010001. However, if odd parity checking is used, then parity bit value is zero; 01010001.

If the original data contains an even number of 1s (1101001), then parity bit of value 1 is added to the data's left side to make the number of 1s odd, if odd parity checking is used and data transmitted becomes 11101001. In case data is transmitted incorrectly, the parity bit value becomes incorrect; thus, indicating error has occurred during transmission.

```
#include<stdio.h>
#include<string.h>
void main()
{ int i;
char name[200];
int one=0,zero=0;
int count=0;
int no;
printf("Enter The Name:-");
gets(name);
printf("Enter The Parity:-");
scanf("%d",&no);
for(i=0;name[i]!='\0';i++)
{
if(name[i]=='1')
{
one++;
}
else
{
zero++;
}
count++;
}
printf("\nZero Are:-%d",zero);
printf("\nOne Are:-%d",one);
if(no==0)
{
if(one%2==0)
{
printf("\nEven Parity");
```

```
                                                  printf("\n0");
                                                 puts(name);
                                                          }
                                                        else
                                                          {
                          printf("\nEven                Parity");
                                                  printf("\n1");
                                                  puts(name);
                                                          }
                                                          }
                                                        else

                                                          {
                                                 if(one%2==0)
                                  {
                          printf("\nOdd                 Parity");
                                                  printf("\n1");
                                                 puts(name);
                                                          }
                                                        else
                                                          {
                          printf("\nOdd                 Parity");
                                                  printf("\n0");
                                                  puts(name);
                                                          }
                                                        }
}
```

ii) AIM: C code to implement multi dimensional parity check

```
#include<iostream>
#include<stdlib.h>
using namespace std;
#define maxlength 10
#define maxmessages 10
void initialize(int arr[][10],int m,int n)
{
for(int i =0;i<m;i++)
for(int j=0;j<n;j++)
{
arr[i][j] = rand()%2;
}
}
void print(int arr[][10],int m,int n)
{
for(int i =0;i<m;i++)
```

```
{  for(int j=0;j<n;j++)
{
cout<<arr[i][j]<<" ";
}
cout<<endl;
}
}
void addparbit(int arr[][10],int m,int n)  // Even Parity
{
for(int i=0;i<m;i++)
{
int count = 0;
for(int j=0;j<n;j++)
{
if(arr[i][j] == 1)
count++;
}
if(count%2 == 0)
arr[i][n] = 0;
else
arr[i][n] = 1;
}
}
void induceerror(int arr[][10],int m,int n)
{
int k1,k2;
k1= rand()%m;
k2 = rand()%n;
if(arr[k1][k2]==0)
arr[k1][k2]=1;
else
arr[k1][k2]=0;
cout<<"Inducing error at line : "<<k1<<endl;
}
void checkerror(int arr[][10],int m,int n)
{
for(int i=0;i<m;i++)
{
int count = 0;
for(int j=0;j<n;j++)
{
if(arr[i][j] == 1)
count++;
}
if(count%2 == 0 && arr[i][n] != 0)
{
cout<<"Error here at line : " <<i;
}
else if(count%2 == 1 && arr[i][n] != 1)
{
cout<<"Error here at line : " <<i;
}

}
}

int main()
{   int m,n,arr[maxmessages][maxlength];
cout<<"Enter total number of messages";
cin>>m;
cout<<"Enter length of each message";
cin>>n;
initialize(arr,m,n);
print(arr,m,n);
addparbit(arr,m,n);
print(arr,m,n+1);
induceerror(arr,m,n);
print(arr,m,n+1);
checkerror(arr,m,n);
return 0;
}
```

6) AIM: C code to implement even odd parity

Procedure :

EVEN AND ODD PARITY

Even parity refers to a parity checking mode in asynchronous communication systems in which an extra bit, called a parity bit, is set to one if there is an even number of one bits in a one-byte data item. If the number of one bits adds up to an odd number, the parity bit is set to zero.

Even parity checking may also be used in testing memory storage devices.

In asynchronous communication systems, odd parity refers to parity checking modes, where each set of transmitted bits has an odd number of bits. If the total number of ones in the data plus the parity bit is an odd number of ones, it is called odd parity. If the data already has an odd number of ones, the value of the added parity bit is 0, otherwise it is 1.

Parity bits are the simplest form of error detection. Odd parity checking is used in testing memory storage devices. The sender and receiver should agree to the use odd parity checking. Without this, successful communication is not possible. If an odd number of bits are switched during transmission, parity checks can detect that the data is corrupted. However, the method

will fail to detect errors introduced when an even number of bits in the same data unit is altered, as the parity will still remain odd despite data.
Parity bits are added to transmitted messages to ensure that the number of bits with a value of one in a set of bits add up to even or odd numbers. Even and odd parities are the two variants of parity checking modes.
Odd parity can be more clearly explained through an example. Consider the transmitted message 1010001, which has three ones in it. This is turned into odd parity by adding a zero, making the sequence 0 1010001. Thus, the total number of ones remain at three, an odd number. If the transmitted message has the form 1101001, which has four ones in it, this can be turned into odd parity by adding a one, making the sequence 1 1101001.
Code :

```
# include <stdio.h>
# define  bool int

/* Function to get parity of number n. It returns 1
   if n has odd parity, and returns 0 if n has even
   parity */
bool getParity(unsigned int n)
{
    bool parity = 0;
    while (n)
    {
        parity = !parity;
        n      = n & (n - 1);
    }
    return parity;
}

/* Driver program to test getParity() */
int main()
{
    unsigned int n = 7;
    printf("Parity of no %d = %s",  n,
             (getParity(n)? "odd": "even"));

    getchar();
    return 0;
}
```

7) Implement data link protocols:
i) AIM: C code to implement unrestricted simplex protocol
procedure :

In order to appreciate the step by step development of efficient and complex protocols such as SDLC, HDLC etc., we will begin with a simple but unrealistic protocol. In this protocol:
Data are transmitted in one direction only
The transmitting (Tx) and receiving (Rx) hosts are always ready
Processing time can be ignored
Infinite buffer space is available
No errors occur; i.e. no damaged frames and no lost frames (perfect channel)

[HEADER.H]

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
typedef struct
{
 int seqno;
 int ackno;
 char data[50];
}frame;
void from_network_layer(char buffer[])
{
 printf("Enter Data : ");
 scanf("%s",buffer);
}
void to_physical_layer(int pid1,frame *f)
{
 write(pid1,f,sizeof(frame));
}
void from_physical_layer(int pid1,frame *f)
{
 read(pid1,f,sizeof(frame));
}
void to_network_layer(char buffer[])
{
 printf("\n%s",buffer);
}
```

[SENDER SIDE]

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
#include                                                    "header.h"
```

```
void main()
{
 int pid1,i,no;
 char buffer[50];
 frame f;
 system(">pipe1");
 pid1=open("pipe1",O_WRONLY);
 printf("Enter NUMBER OF DATA : ");
 scanf("%d",&no);
 write(pid1,&no,sizeof(no));
 for(i=0;i<no;i++)
  {
  from_network_layer(buffer);
  strcpy(f.data,buffer);
  to_physical_layer(pid1,&f);
  }
 close(pid1);
}
```

[RECEIVER SIDE]

```
#include<stdio.h>
#include<fcntl.h>
#include<string.h>
#include "header.h"
void main()
{
 int pid1,no,i;
 char buffer[50];
 frame f;
 pid1=open("pipe1",O_RDONLY);
 read(pid1,&no,sizeof(no));
 printf("DATA RECEIVED : %d",no);
 printf("\nDATA");
 for(i=0;i<no;i++)
  {
  from_physical_layer(pid1,&f);
  strcpy(buffer,f.data);
  to_network_layer(buffer);
  }
 close(pid1);
 unlink("pipe1");
}
```

ii)

AIM: C code to generate stop and wait protocol

Procedure :

Stop-and-wait ARQ, also referred to as alternating bit protocol, is a method in telecommunications to send information between two connected devices. It ensures that information is not lost due to dropped packets and that packets are received in the correct order. It is the simplest automatic repeat-request (ARQ) mechanism. A stop-and-wait ARQ sender sends one frame at a time; it is a special case of the general sliding window protocol with transmit and receive window sizes equal to one and greater than one respectively. After sending each frame, the sender doesn't send any further frames until it receives an acknowledgement (ACK) signal. After receiving a valid frame, the receiver sends an ACK. If the ACK does not reach the sender before a certain time, known as the timeout, the sender sends the same frame again. The timeout countdown is reset after each frame transmission. The above behavior is a basic example of Stop-and-Wait. However, real-life implementations vary to address certain issues of design.

Typically the transmitter adds a redundancy check number to the end of each frame. The receiver uses the redundancy check number to check for possible damage. If the receiver sees that the frame is good, it sends an ACK. If the receiver sees that the frame is damaged, the receiver discards it and does not send an ACK—pretending that the frame was completely lost, not merely damaged.

One problem is when the ACK sent by the receiver is damaged or lost. In this case, the sender doesn't receive the ACK, times out, and sends the frame again. Now the receiver has two copies of the same frame, and doesn't know if the second one is a duplicate frame or the next frame of the sequence carrying identical data.

Another problem is when the transmission medium has such a long latency that the sender's timeout runs out before the frame reaches the receiver. In this case the sender resends the same packet. Eventually the receiver gets two copies of the same frame, and sends an ACK for each one. The sender, waiting for a single ACK, receives two ACKs, which may cause problems if it assumes that the second ACK is for the next frame in the sequence.

To avoid these problems, the most common solution is to define a 1 bit sequence number in the header of the frame. This sequence number alternates (from 0 to 1) in subsequent frames. When the receiver sends an ACK, it includes the sequence number of the next packet it expects. This way, the receiver can detect duplicated frames by checking if the frame sequence numbers alternate. If two subsequent frames have the same sequence number, they are duplicates, and the second frame is discarded. Similarly, if two subsequent ACKs reference the same sequence number, they are acknowledging the same frame.

Stop-and-wait ARQ is inefficient compared to other ARQs, because the time between packets, if the ACK and the data are received successfully, is twice the transit time (assuming the turnaround time can be zero). The throughput on the channel is a fraction of what it could be. To solve this problem, one can send more than one packet at a time with a larger sequence number and use one ACK for a set. This is what is done in Go-Back-N ARQ and the Selective Repeat ARQ.

Code :

```
#include <cnet.h>
```

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

/*  This is an implementation of a stop-and-wait data link protocol.
    It is based on Tanenbaum's `protocol 4', 2nd edition, p227
    (or his 3rd edition, p205).
    This protocol employs only data and acknowledgement frames -
    piggybacking and negative acknowledgements are not used.

    It is currently written so that only one node (number 0) will
    generate and transmit messages and the other (number 1) will receive
    them. This restriction seems to best demonstrate the protocol to
    those unfamiliar with it.
    The restriction can easily be removed by "commenting out" the line

            if(nodeinfo.nodenumber == 0)

    in reboot_node(). Both nodes will then transmit and receive (why?).

    Note that this file only provides a reliable data-link layer for a
    network of 2 nodes.
 */


typedef enum    { DL_DATA, DL_ACK }   FRAMEKIND;

typedef struct {
    char        data[MAX_MESSAGE_SIZE];
} MSG;

typedef struct {
    FRAMEKIND    kind;          /* only ever DL_DATA or DL_ACK */
    unsigned int len;           /* the length of the msg field only */
    int          checksum;      /* checksum of the whole frame */
    int          seq;           /* only ever 0 or 1 */
    MSG          msg;
} FRAME;

#define FRAME_HEADER_SIZE  (sizeof(FRAME) - sizeof(MSG))
#define FRAME_SIZE(f)      (FRAME_HEADER_SIZE + f.len)


static  MSG             *lastmsg;
static  unsigned int    lastlength          = 0;
static  CnetTimerID     lasttimer           = NULLTIMER;
static  int          ackexpected          = 0;
static  int          nextframetosend       = 0;
static  int          frameexpected         = 0;


static void transmit_frame(MSG *msg, FRAMEKIND kind,
                    unsigned int length, int seqno)
{
    FRAME       f;
    int         link = 1;

    f.kind      = kind;
    f.seq       = seqno;
    f.checksum  = 0;
    f.len       = length;

    switch (kind) {
    case DL_ACK :
       printf("ACK transmitted, seq=%d\n", seqno);
       break;

    case DL_DATA: {
       CnetTime        timeout;

       printf(" DATA transmitted, seq=%d\n", seqno);
       memcpy(&f.msg, (char *)msg, (int)length);

       timeout = FRAME_SIZE(f)*((CnetTime)8000000 / linkinfo[link].bandwidth) +
                       linkinfo[link].propagationdelay;

       lasttimer = CNET_start_timer(EV_TIMER1, 3 * timeout, 0);
       break;
      }
    }
    length      = FRAME_SIZE(f);
    f.checksum  = CNET_ccitt((unsigned char *)&f, (int)length);
    CHECK(CNET_write_physical(link, (char *)&f, &length));
}


static void application_ready(CnetEvent ev, CnetTimerID timer, CnetData data)
{
    CnetAddr destaddr;
```

```
    lastlength  = sizeof(MSG);
    CHECK(CNET_read_application(&destaddr, (char *)lastmsg, &lastlength));
    CNET_disable_application(ALLNODES);

    printf("down from application, seq=%d\n", nextframetosend);
    transmit_frame(lastmsg, DL_DATA, lastlength, nextframetosend);
    nextframetosend = 1-nextframetosend;
}


static void physical_ready(CnetEvent ev, CnetTimerID timer, CnetData data)
{
    FRAME       f;
    unsigned int len;
    int         link, checksum;

    len         = sizeof(FRAME);
    CHECK(CNET_read_physical(&link, (char *)&f, &len));

    checksum    = f.checksum;
    f.checksum  = 0;
    if(CNET_ccitt((unsigned char *)&f, (int)len) != checksum) {
        printf("\t\t\t\tBAD checksum - frame ignored\n");
        return;           /* bad checksum, ignore frame */
    }

    switch (f.kind) {
    case DL_ACK :
        if(f.seq == ackexpected) {
            printf("\t\t\t\tACK received, seq=%d\n", f.seq);
            CNET_stop_timer(lasttimer);
            ackexpected = 1-ackexpected;
            CNET_enable_application(ALLNODES);
        }
        break;

    case DL_DATA :
        printf("\t\t\t\tDATA received, seq=%d, ", f.seq);
        if(f.seq == frameexpected) {
            printf("up to application\n");
            len = f.len;
            CHECK(CNET_write_application((char *)&f.msg, &len));
            frameexpected = 1-frameexpected;
        }
        else
```

```
          printf("ignored\n");
        transmit_frame((MSG *)NULL, DL_ACK, 0, f.seq);
        break;
    }
}


static void draw_frame(CnetEvent ev, CnetTimerID timer, CnetData data)
{
    CnetDrawFrame *df   = (CnetDrawFrame *)data;
    FRAME         *f    = (FRAME *)df->frame;

    switch (f->kind) {
    case DL_ACK :
        df->colour[0]   = (f->seq == 0) ? CN_RED : CN_PURPLE;
        df->pixels[0]   = 10;
        sprintf(df->text, "%d", f->seq);
        break;

    case DL_DATA :
        df->colour[0]   = (f->seq == 0) ? CN_RED : CN_PURPLE;
        df->pixels[0]   = 10;
        df->colour[1]   = CN_GREEN;
        df->pixels[1]   = 30;
        sprintf(df->text, "data=%d", f->seq);
        break;
    }
}


static void timeouts(CnetEvent ev, CnetTimerID timer, CnetData data)
{
    if(timer == lasttimer) {
        printf("timeout, seq=%d\n", ackexpected);
        transmit_frame(lastmsg, DL_DATA, lastlength, ackexpected);
    }
}


static void showstate(CnetEvent ev, CnetTimerID timer, CnetData data)
{
    printf(
    "\n\tackexpected\t= %d\n\tnextframetosend\t= %d\n\tframeexpected\t= %d\n",
                ackexpected, nextframetosend, frameexpected);
}
```

```
void reboot_node(CnetEvent ev, CnetTimerID timer, CnetData data)
{
   if(nodeinfo.nodenumber > 1) {
      fprintf(stderr,"This is not a 2-node network!\n");
      exit(1);
   }

   lastmsg     = malloc(sizeof(MSG));

   CHECK(CNET_set_handler( EV_APPLICATIONREADY, application_ready, 0));
   CHECK(CNET_set_handler( EV_PHYSICALREADY,    physical_ready, 0));
   CHECK(CNET_set_handler( EV_DRAWFRAME,        draw_frame, 0));
   CHECK(CNET_set_handler( EV_TIMER1,           timeouts, 0));
   CHECK(CNET_set_handler( EV_DEBUG0,           showstate, 0));

   CHECK(CNET_set_debug_string( EV_DEBUG0, "State"));

   if(nodeinfo.nodenumber == 1)
      CNET_enable_application(ALLNODES);
}
```

iii) Selective repeat sliding window protocol

Aim: To implement Selective window sliding protocol

Procedure :

Transmitter operation

Whenever the transmitter has data to send, it may transmit up to w_t packets ahead of the latest acknowledgment n_a. That is, it may transmit packet number n_t as long as $n_t < n_a+w_t$.

In the absence of a communication error, the transmitter soon receives an acknowledgment for all the packets it has sent, leaving n_a equal to n_t. If this does not happen after a reasonable delay, the transmitter must retransmit the packets between n_a and n_t.

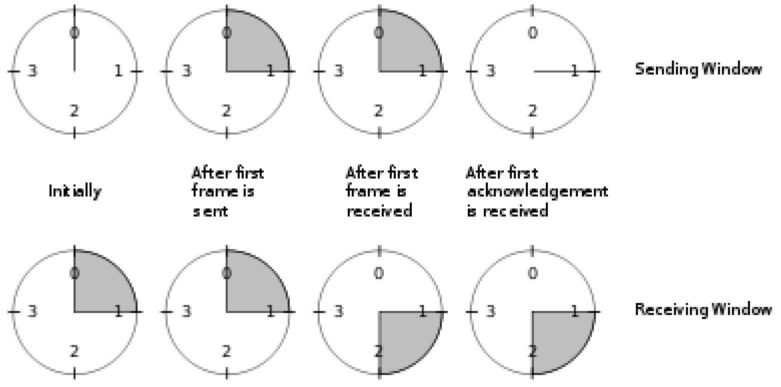
Techniques for defining "reasonable delay" can be extremely elaborate, but they only affect efficiency; the basic reliability of the sliding window protocol does not depend on the details.

Receiver operation

Every time a packet numbered x is received, the receiver checks to see if it falls in the receive window, $n_r \leq x < n_r+w_r$. (The simplest receivers only have to keep track of one value $n_r=n_s$.) If it falls within the window, the receiver accepts it. If it is numbered n_r, the receive sequence number is increased by 1, and possibly more if further consecutive packets were previously received and stored. If $x > n_r$, the packet is stored until all preceding packets have been received.[1] If $x \geq n_s$, the latter is updated to $n_s=x+1$.

If the packet's number is not within the receive window, the receiver discards it and does not modify n_r or n_s.

Whether the packet was accepted or not, the receiver transmits an acknowledgment containing the current n_r. (The acknowledgment may also include information about additional packets received between n_r or n_s, but that only helps efficiency.)
Note that there is no point having the receive window w_r larger than the transmit window w_t, because there is no need to worry about receiving a packet that will never be transmitted; the useful range is $1 \leq w_r \leq w_t$.
Sequence number range required
Main article: serial number arithmetic


Sending Window
Initially
After first frame is sent
After first frame is received
After first acknowledgement is received
Receiving Window
0
1
2
3

A sliding window with a 2-bit sequence, of size 1

Sequence numbers modulo 4, with w_r=1. Initially, n_t=n_r=0
So far, the protocol has been described as if sequence numbers are of unlimited size, ever-increasing. However, rather than transmitting the full sequence number x in messages, it is possible to transmit only x mod N, for some finite N. (N is usually a power of 2.)
For example, the transmitter will only receive acknowledgments in the range n_a to n_t, inclusive. Since it guarantees that $n_t - n_a \leq w_t$, there are at most w_t+1 possible sequence numbers that could arrive at any given time. Thus, the transmitter can unambiguously decode the sequence number as long as $N > w_t$.
A stronger constraint is imposed by the receiver. The operation of the protocol depends on the receiver being able to reliably distinguish new packets (which should be accepted and processed) from retransmissions of old packets (which should be discarded, and the last acknowledgment retransmitted). This can be done given knowledge of the transmitter's window size. After receiving a packet numbered x, the receiver knows that $x < n_a+w_t$, so $n_a > x-w_t$. Thus, packets numbered $x-w_t$ will never again be retransmitted.
The lowest sequence number we will ever receive in future is n_s-w_t
The receiver also knows that the transmitter's n_a cannot be higher than the highest acknowledgment ever sent, which is n_r. So the highest sequence number we could possibly see is $n_r+w_t \leq n_s+w_t$.
Thus, there are $2w_t$ different sequence numbers that the receiver can receive at any one time. It might therefore seem that we must have $N \geq 2w_t$. However, the actual limit is lower.
The additional insight is that the receiver does not need to distinguish between sequence numbers that are too low (less than n_r) or that are too high (greater than or equal to n_s+w_r). In either case, the receiver ignores the packet except to retransmit an acknowledgment. Thus, it is only

necessary that $N \geq w_t+w_r$. As it is common to have $w_r<w_t$ (e.g. see Go-Back-Nbelow), this can permit larger w_t within a fixed N.
Code :

```
#include<stdio.h>
#include<stdlib.h>
main()
int i,m,n,j,w,1;
char c;
FILE*f;
f=fopen("text.txt","r");
printf("window size");
scanf("%d",&n);
m=n;
while(!fof(f))
{
i=rand()%n+1;
j=i;
1=i;
if(m>i)
{
m=m-i;
if(m>0)
{
printf("\n");
while(i>0 & !feof(f))
{
c=getc(f);
printf("%c",c);
i--;
}
printf("\n%d transferred"j);
if(j>3)
printf("\n 1 acknowledgement received");
else
printf("\n acknowledgement received",j+1);
}
} m=m+j-1;}
```

8. Implement i. Message Authentication Codes

ii. Cryptographic Hash Functions and Applications.

i. Message Authentication Codes

```
#include "core/crypto.h"
 #include "mac/cmac.h"

 //Check crypto library configuration
 #if (CMAC_SUPPORT == ENABLED)


 /**
  * @brief Compute CMAC using the specified cipher algorithm
  * @param[in] cipher Cipher algorithm used to compute CMAC
  * @param[in] key Pointer to the secret key
  * @param[in] keyLen Length of the secret key
  * @param[in] data Pointer to the input message
  * @param[in] dataLen Length of the input data
  * @param[out] mac Calculated MAC value
  * @param[in] macLen Expected length of the MAC
  * @return Error code
  **/

 error_t cmacCompute(const CipherAlgo *cipher, const void *key, size_t keyLen,
    const void *data, size_t dataLen, uint8_t *mac, size_t macLen)
 {
    error_t error;
    CmacContext *context;

    //Allocate a memory buffer to hold the CMAC context
    context = cryptoAllocMem(sizeof(CmacContext));

    //Successful memory allocation?
    if(context != NULL)
    {
       //Initialize the CMAC context
       error = cmacInit(context, cipher, key, keyLen);

       //Check status code
       if(!error)
       {
          //Digest the message
          cmacUpdate(context, data, dataLen);
          //Finalize the CMAC computation
          error = cmacFinal(context, mac, macLen);
```

```
      }

      //Free previously allocated memory
      cryptoFreeMem(context);
   }
   else
   {
      //Failed to allocate memory
      error = ERROR_OUT_OF_MEMORY;
   }

   //Return status code
   return error;
}


/**
 * @brief Initialize CMAC calculation
 * @param[in] context Pointer to the CMAC context to initialize
 * @param[in] cipher Cipher algorithm used to compute CMAC
 * @param[in] key Pointer to the secret key
 * @param[in] keyLen Length of the secret key
 * @return Error code
 **/

error_t cmacInit(CmacContext *context, const CipherAlgo *cipher,
   const void *key, size_t keyLen)
{
   error_t error;
   uint8_t rb;

   //CMAC supports only block ciphers whose block size is 64 or 128 bits
   if(cipher->type != CIPHER_ALGO_TYPE_BLOCK)
      return ERROR_INVALID_PARAMETER;

   //Rb is completely determined by the number of bits in a block
   if(cipher->blockSize == 8)
   {
      //If b = 64, then Rb = 11011
      rb = 0x1B;
   }
   else if(cipher->blockSize == 16)
   {
      //If b = 128, then Rb = 10000111
      rb = 0x87;
```

```
  }
  else
  {
    //Invalid block size
    return ERROR_INVALID_PARAMETER;
  }

  //Cipher algorithm used to compute CMAC
  context->cipher = cipher;

  //Initialize cipher context
  error = cipher->init(context->cipherContext, key, keyLen);
  //Any error to report?
  if(error)
    return error;

  //Let L = 0
  cryptoMemset(context->buffer, 0, cipher->blockSize);
  //Compute L = CIPH(L)
  cipher->encryptBlock(context->cipherContext, context->buffer, context->buffer);

  //The subkey K1 is obtained by multiplying L by x in GF(2^b)
  cmacMul(context->k1, context->buffer, cipher->blockSize, rb);
  //The subkey K2 is obtained by multiplying L by x^2 in GF(2^b)
  cmacMul(context->k2, context->k1, cipher->blockSize, rb);

  //Reset CMAC context
  cmacReset(context);

  //Successful initialization
  return NO_ERROR;
}


/**
 * @brief Reset CMAC context
 * @param[in] context Pointer to the CMAC context
 **/

void cmacReset(CmacContext *context)
{
  //Clear input buffer
  cryptoMemset(context->buffer, 0, context->cipher->blockSize);
  //Number of bytes in the buffer
  context->bufferLength = 0;
```

```
   //Initialize MAC value
   cryptoMemset(context->mac, 0, context->cipher->blockSize);
}


/**
 * @brief Update the CMAC context with a portion of the message being hashed
 * @param[in] context Pointer to the CMAC context
 * @param[in] data Pointer to the input data
 * @param[in] dataLen Length of the buffer
 **/

void cmacUpdate(CmacContext *context, const void *data, size_t dataLen)
{
   size_t n;

   //Process the incoming data
   while(dataLen > 0)
   {
      //Process message block by block
      if(context->bufferLength == context->cipher->blockSize)
      {
         //XOR M(i) with C(i-1)
         cmacXorBlock(context->buffer, context->buffer, context->mac,
            context->cipher->blockSize);

         //Compute C(i) = CIPH(M(i) ^ C(i-1))
         context->cipher->encryptBlock(context->cipherContext, context->buffer,
            context->mac);

         //Empty the buffer
         context->bufferLength = 0;
      }

      //The message is partitioned into complete blocks
      n = MIN(dataLen, context->cipher->blockSize - context->bufferLength);

      //Copy the data to the buffer
      cryptoMemcpy(context->buffer + context->bufferLength, data, n);
      //Update the length of the buffer
      context->bufferLength += n;

      //Advance the data pointer
      data = (uint8_t *) data + n;
      //Remaining bytes to process
      dataLen -= n;
   }
}


/**
 * @brief Finish the CMAC calculation
 * @param[in] context Pointer to the CMAC context
 * @param[out] mac Calculated MAC value
 * @param[in] macLen Expected length of the MAC
 * @return Error code
 **/

error_t cmacFinal(CmacContext *context, uint8_t *mac, size_t macLen)
{
   //Check the length of the MAC
   if(macLen < 1 || macLen > context->cipher->blockSize)
      return ERROR_INVALID_PARAMETER;

   //Check whether the last block Mn is complete
   if(context->bufferLength >= context->cipher->blockSize)
   {
      //The final block M(n) is XOR-ed with the first subkey K1
      cmacXorBlock(context->buffer, context->buffer, context->k1,
         context->cipher->blockSize);
   }
   else
   {
      //Append padding string
      context->buffer[context->bufferLength++] = 0x80;

      //Append the minimum number of zeroes to form a complete block
      while(context->bufferLength < context->cipher->blockSize)
         context->buffer[context->bufferLength++] = 0x00;

      //The final block M(n) is XOR-ed with the second subkey K2
      cmacXorBlock(context->buffer, context->buffer, context->k2,
         context->cipher->blockSize);
   }

   //XOR M(n) with C(n-1)
   cmacXorBlock(context->buffer, context->buffer, context->mac,
      context->cipher->blockSize);
```

```
  //Compute T = CIPH(M(n) ^ C(n-1))
  context->cipher->encryptBlock(context->cipherContext, context->buffer,
    context->mac);

  //Copy the resulting MAC value
  if(mac != NULL)
  {
    //It is possible to truncate the MAC. The result of the truncation
    //should be taken in most significant bits first order
    cryptoMemcpy(mac, context->mac, macLen);
  }

  //Successful processing
  return NO_ERROR;
}


/**
 * @brief Multiplication by x in GF(2^128)
 * @param[out] x Pointer to the output block
 * @param[out] a Pointer to the input block
 * @param[in] n Size of the block, in bytes
 * @param[in] rb Representation of the irreducible binary polynomial
 **/

void cmacMul(uint8_t *x, const uint8_t *a, size_t n, uint8_t rb)
{
  size_t i;
  uint8_t c;

  //Save the value of the most significant bit
  c = a[0] >> 7;

  //The multiplication of a polynomial by x in GF(2^128) corresponds to a
  //shift of indices
  for(i = 0; i < (n - 1); i++)
  {
    x[i] = (a[i] << 1) | (a[i + 1] >> 7);
  }

  //Shift the last byte of the block to the left
  x[i] = a[i] << 1;

  //If the highest term of the result is equal to one, then perform reduction
  x[i] ^= rb & ~(c - 1);
}


/**
 * @brief XOR operation
 * @param[out] x Block resulting from the XOR operation
 * @param[in] a First input block
 * @param[in] b Second input block
 * @param[in] n Size of the block, in bytes
 **/

void cmacXorBlock(uint8_t *x, const uint8_t *a, const uint8_t *b, size_t n)
{
  size_t i;

  //Perform XOR operation
  for(i = 0; i < n; i++)
  {
    x[i] = a[i] ^ b[i];
  }
}

#endif
```

ii. Cryptographic Hash Functions and Applications.

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

typedef union uwb {
   unsigned w;
   unsigned char b[4];
} MD5union;

typedef unsigned DigestArray[4];

unsigned func0( unsigned abcd[] ){
   return ( abcd[1] & abcd[2]) | (~abcd[1] & abcd[3]);}

unsigned func1( unsigned abcd[] ){
   return ( abcd[3] & abcd[1]) | (~abcd[3] & abcd[2]);}

unsigned func2( unsigned abcd[] ){
   return  abcd[1] ^ abcd[2] ^ abcd[3];}
```

```
unsigned func3( unsigned abcd[] ){
   return abcd[2] ^ (abcd[1] |~ abcd[3]);}

typedef unsigned (*DgstFctn)(unsigned a[]);

unsigned *calctable( unsigned *k)
{
   double s, pwr;
   int i;

   pwr = pow( 2, 32);
   for (i=0; i<64; i++) {
      s = fabs(sin(1+i));
      k[i] = (unsigned)( s * pwr );
   }
   return k;
}

/*Rotate Left r by N bits
 or
We can directly hardcode below table but as i explained above we are opting
generic code so shifting the bit manually.
r[ 0..15] := {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22}
r[16..31] := {5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20, 5,  9, 14, 20}
r[32..47] := {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23}
r[48..63] := {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}
*/
unsigned rol( unsigned r, short N )
{
   unsigned  mask1 = (1<<N) -1;
   return ((r>>(32-N)) & mask1) | ((r<<N) & ~mask1);
}

unsigned *md5( const char *msg, int mlen)
{
   /*Initialize Digest Array as A , B, C, D */
   static DigestArray h0 = { 0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476 };
   static DgstFctn ff[] = { &func0, &func1, &func2, &func3 };
   static short M[] = { 1, 5, 3, 7 };
   static short O[] = { 0, 1, 5, 0 };
   static short rot0[] = { 7,12,17,22};
   static short rot1[] = { 5, 9,14,20};
   static short rot2[] = { 4,11,16,23};
   static short rot3[] = { 6,10,15,21};
   static short *rots[] = {rot0, rot1, rot2, rot3 };
   static unsigned kspace[64];
   static unsigned *k;

   static DigestArray h;
   DigestArray abcd;
   DgstFctn fctn;
   short m, o, g;
   unsigned f;
   short *rotn;
   union {
      unsigned w[16];
      char     b[64];
   }mm;
   int os = 0;
   int grp, grps, q, p;
   unsigned char *msg2;

   if (k==NULL) k= calctable(kspace);

   for (q=0; q<4; q++) h[q] = h0[q];   // initialize

   {
      grps  = 1 + (mlen+8)/64;
      msg2 = malloc( 64*grps);
      memcpy( msg2, msg, mlen);
      msg2[mlen] = (unsigned char)0x80;
      q = mlen + 1;
      while (q < 64*grps){ msg2[q] = 0; q++ ; }
      {
         MD5union u;
         u.w = 8*mlen;
         q -= 8;
         memcpy(msg2+q, &u.w, 4 );
      }
   }

   for (grp=0; grp<grps; grp++)
   {
      memcpy( mm.b, msg2+os, 64);
      for(q=0;q<4;q++) abcd[q] = h[q];
      for (p = 0; p<4; p++) {
         fctn = ff[p];
         rotn = rots[p];
         m = M[p]; o= O[p];
```

```
        for (q=0; q<16; q++) {
            g = (m*q + o) % 16;
            f = abcd[1] + rol( abcd[0]+ fctn(abcd) + k[q+16*p] + mm.w[g], rotn[q%4]);

            abcd[0] = abcd[3];
            abcd[3] = abcd[2];
            abcd[2] = abcd[1];
            abcd[1] = f;
        }
    }
    for (p=0; p<4; p++)
        h[p] += abcd[p];
    os += 64;
  }
  return h;
}

int main( int argc, char *argv[] )
{
  int j,k;
  const char *msg = "This code has been provided by C codechamp";
  printf("----------------------------------------------------\n");
  printf("-------------Made by C codechamp--------------------\n");
  printf("----------------------------------------------------\n\n");
  printf("\t MD5 ENCRYPTION ALGORITHM IN C \n\n");
  printf("Input String to be Encrypted using MD5 : \n\t%s",msg);
  unsigned *d = md5(msg, strlen(msg));
  MD5union u;
  printf("\n\n\nThe MD5 code for input string is : \n");
  printf("\t= 0x");
  for (j=0;j<4; j++){
    u.w = d[j];
    for (k=0;k<4;k++) printf("%02x",u.b[k]);
  }
  printf("\n");
  printf("\n\t MD5 Encyption Successfully Completed!!!\n\n");
  getch();
  system("pause");
  return 0;
}
```

9. Implement Symmetric Key Encryption Standards (DES) and (AES).

```
#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");
        else // size will contain no. of chars in input file.
                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;
}
```

10. Implement Diffie-Hellman Key Establishment.

```
#include <stdio.h>

// Function to compute a^m mod n
int compute(int a, int m, int n)
{
        int r;
        int y = 1;

        while (m > 0)
        {
                r = m % 2;

                // fast exponention
                if (r == 1)
                        y = (y*a) % n;
                a = a*a % n;

                m = m / 2;
        }

        return y;
}

// C program to demonstrate Diffie-Hellman algorithm
int main()
{
        int p = 23;             // modulus
        int g = 5;              // base

        int a, b;       // a - Alice's Secret Key, b - Bob's Secret Key.
        int A, B;       // A - Alice's Public Key, B - Bob's Public Key

        // choose secret integer for Alice's Pivate Key (only known to Alice)
        a = 6;          // or use rand()

        // Calculate Alice's Public Key (Alice will send A to Bob)
        A = compute(g, a, p);

        // choose secret integer for Bob's Pivate Key (only known to Bob)
```

```
    b = 15;          // or use rand()

    // Calculate Bob's Public Key (Bob will send B to Alice)
    B = compute(g, b, p);

    // Alice and Bob Exchanges their Public Key A & B with each other

    // Find Secret key
    int keyA = compute(B, a, p);
    int keyB = compute(A, b, p);

    printf("Alice's Secret Key is %d\nBob's Secret Key is %d", keyA, keyB);

    return 0;
}
```