Assignment 1

1 Bubble

```
#include <stdio.h>

struct packet
{
   int seqno;
   char a[100];
   float time;
} p[100], temp;

void bubblesort(int n)
{
   int i, j;
   for (i = 0; i < n - 1; i++)
   {
      for (j = 0; j < n - i - 1; j++)
      {
         if (p[j].seqno > p[j + 1].seqno)
         {
            temp = p[j + 1];
            p[j + 1] = p[j];
            p[j] = temp;
         }
      }
   }
}

void main()
{
   int i, j, n, num;
   printf("Enter the number of frames:");
   scanf("%d", &n);
   for (i = 0; i < n; i++)
   {
   slabel:
      num = rand() % 100;
      for (j = 0; j <= i; j++)
      {
         if (p[j].seqno == num)
         {
            goto slabel;
         }
      }
      p[i].seqno = num;
      printf("Enter message:");
      scanf("%s", &p[i].a);
```

```
        p[i].time = rand();
    }

    printf("Before sorting:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%s\t%f\n", p[i].seqno, p[i].a, p[i].time);
    }

    bubblesort(n);
    printf("After sorting:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%s\t%f\n", p[i].seqno, p[i].a, p[i].time);
    }
}
```

2 Insertion

```
#include <stdio.h>

struct packet
{
    int seqno;
    int ponum;
    char sip[20];
    char dip[20];
    int pid;
    char a[100];
    float time;
} p[100], temp;

void insertionsort(int n)
{
    int i, j;
    for (i = 1; i < n - 1; i++)
    {
        temp = p[i];
        for (j = i - 1; j >= 0 && temp.seqno < p[j].seqno; j--)
        {
            p[j + 1] = p[j];
        }
        p[j + 1] = temp;
    }
}

void main()
```

```
{
    int i, j, n, num;
    printf("Enter the number of frames:");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
    slabel:
        num = rand() % 100;
        for (j = 0; j <= i; j++)
        {
            if (p[j].seqno == num)
            {
                goto slabel;
            }
        }
        p[i].seqno = num;
        printf("Enter message:");
        scanf("%s", &p[i].a);
        printf("Enter packet id:");
        scanf("%d", &p[i].pid);
        printf("Enter port number:");
        scanf("%d", &p[i].ponum);
        printf("Enter source ip:");
        scanf("%s", &p[i].sip);
        printf("Enter destination ip:");
        scanf("%s", &p[i].dip);
    }

    printf("Before sorting:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t%d\t%s\t%s\t%s\n", p[i].seqno, p[i].pid, p[i].ponum,
p[i].sip, p[i].dip, p[i].a);
    }

    insertionsort(n);
    printf("After sorting:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%d\t%d\t%s\t%s\t%s\n", p[i].seqno, p[i].pid, p[i].ponum,
p[i].sip, p[i].dip, p[i].a);
    }
}
```

3 quick

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

```
struct packet
{
    int seqno;
    char a[100];
    float time;
} p[100], temp;

void quicksort(int left, int right)
{
    int i, j, pivot;
    if (left < right)
    {
        i = left;
        j = right + 1;
        pivot = p[left].seqno;
        do
        {
            do
            {
                i++;
            }

            while (p[i].seqno < pivot && i <= right);
            do
            {
                j--;
            }

            while (p[j].seqno > pivot && j >= left);
            if (i < j)
            {
                temp = p[i];
                p[i] = p[j];
                p[j] = temp;
            }
        }

        while (i <= j);
        temp = p[left];
        p[left] = p[j];
        p[j] = temp;
        quicksort(left, j - 1);
        quicksort(j + 1, right);
    }
}
```

```
void main()
{
    int i, j, n, num;
    printf("Enter the number of frames:");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
    slabel:
        num = rand() % 100;
        for (j = 0; j < i; j++)
        {
            if (p[j].seqno == num)
                goto slabel;
        }
        p[i].seqno = num;
        printf("Enter message:");
        scanf("%s", p[i].a);
        p[i].time = rand();
    }

    printf("Before sorting:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%s\t%f\n", p[i].seqno, p[i].a, p[i].time);
    }

    quicksort(0, n - 1);
    printf("After sorting:\n");
    for (i = 0; i < n; i++)
    {
        printf("%d\t%s\t%f\n", p[i].seqno, p[i].a, p[i].time);
    }
}
```

Assignment 2

1 4 nodes

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf

proc finish {} {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit(0)
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 100Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 5ms DropTail
$ns duplex-link $n2 $n3 1Mb 5ms DropTail
$ns duplex-link $n1 $n3 1Mb 5ms DropTail

$ns queue-limit $n0 $n2 50
$ns queue-limit $n1 $n2 50
$ns queue-limit $n2 $n3 50
$ns queue-limit $n1 $n3 50

set udp1 [new Agent/UDP]
$ns attach-agent $n0 $udp1
set null [new Agent/Null]
$ns attach-agent $n3 $null

$ns connect $udp1 $null
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packet-size 200
$cbr1 set interval 0.005
$cbr1 attach-agent $udp1

$ns at 0.5 "$cbr1 start"
$ns at 5.0 "$cbr1 stop"
$ns at 5.5 "finish"
$ns run
```

2 6 nodes

```
set ns [new Simulator]
$ns color 1 violet
set nf [open out.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf

proc finish {} {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit(0)
}

set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$ns duplex-link $n1 $n5 0.011Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 5ms DropTail
$ns duplex-link $n2 $n4 1Mb 5ms DropTail
$ns duplex-link $n1 $n6 1Mb 5ms DropTail
$ns duplex-link $n3 $n6 1Mb 5ms DropTail

$ns duplex-link-op $n1 $n5 color red
$ns duplex-link-op $n2 $n3 color blue

$ns duplex-link-op $n1 $n5 orient left
$ns duplex-link-op $n2 $n3 orient right

$ns queue-limit $n1 $n5 50
$ns queue-limit $n2 $n3 0.011
$ns queue-limit $n2 $n4 50
$ns queue-limit $n1 $n6 50
$ns queue-limit $n3 $n6 50

$n1 color blue
$n2 color red

$n1 shape circle
$n2 shape box

set tcp1 [new Agent/TCP]
$ns attach-agent $n3 $tcp1
$tcp1 set fid_ 1
set tcpsink0 [new Agent/TCPSink]
$ns  attach-agent $n5 $tcpsink0
$ns connect $tcp1 $tcpsink0

set tcp2 [new Agent/TCP]
$ns attach-agent $n4 $tcp2
```

```
set tcpsink1 [new Agent/TCPSink]
$ns attach-agent $n6 $tcpsink1
$ns connect $tcp2 $tcpsink1

set ftp0 [new Application/FTP]
set telnet0 [new Application/Telnet]

$ftp0 set packet-size 200
$ftp0 set Interval 0.005
$ftp0 attach-agent $tcp1
$ns at 0.5 "$ftp0 start"
$ns at 4.5 "$ftp0 stop"

$telnet0 set packet-size 200
$telnet0 set interval 0.005
$telnet0 attach-agent $tcp2
$ns at 0.5 "$telnet0 start"
$ns at 4.5 "$telnet0 stop"

$ns at 5.5 "finish"
$ns run
```

Assignment 3

1 10 node

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf

$ns color 1 red
$ns color 2 brown

proc finish {} {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit(0)
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]

$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n5 $n6 $n7 $n8 $n9"  0.1Mb 0.1ms LL
Queue/DropTail Mac/802_3

set tcp0 [new Agent/TCP]
$ns attach-agent $n3 $tcp0
$tcp0 set fid_ 1
set tcpsink0 [new Agent/TCPSink]
$ns attach-agent $n5 $tcpsink0
$ns connect $tcp0 $tcpsink0

set telnet0 [new Application/Telnet]
$telnet0 set packet-size 500
$telnet0 set interval 0.005
$telnet0 attach-agent $tcp0
$ns at 0.5 "$telnet0 start"
$ns at 2.0 "$telnet0 stop"

set tcp1 [new Agent/TCP]
$ns attach-agent $n3 $tcp1
$tcp1 set fid_ 2
set tcpsink0 [new Agent/TCPSink]
$ns attach-agent $n5 $tcpsink0
$ns connect $tcp1 $tcpsink0

set telnet1 [new Application/Telnet]
$telnet1 set packet-size 500
$telnet1 set interval 0.005
$telnet1 attach-agent $tcp1
$ns at 0.5 "$telnet1 start"
$ns at 2.0 "$telnet1 stop"

$ns at 5.0 "finish"
$ns run
```

2 12 node

```
set ns [new Simulator]
$ns color 1 brown
$ns color 2 green
set nf [open out.nam w]
$ns namtrace-all $nf
```

```
set tf [open out.tr w]
$ns trace-all $tf

proc finish {} {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit (0)
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
set n9 [$ns node]
set n10 [$ns node]
set n11 [$ns node]

$ns make-lan "$n0 $n1 $n2 $n3 $n4 $n5 $n6 $n7 $n8 $n9 $n10 $n11"  0.22Mb
10ms LL Queue/DropTail Mac/802_3

$n0 color red
$n1 color red
$n2 color red
$n3 color red
$n4 color blue
$n5 color blue
$n6 color blue
$n7 color blue
$n8 color blue
$n9 color purple
$n10 color purple
$n11 color purple

set udp1 [new Agent/UDP]
$ns attach-agent $n0 $udp1
set null1 [new Agent/Null]
$ns attach-agent $n3 $null1
$ns connect $udp1 $null1
```

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n4 $tcp1
$tcp1 set fid_ 1
set tcpsink0 [new Agent/TCPSink]
$ns attach-agent $n8 $tcpsink0
$ns connect $tcp1 $tcpsink0

set tcp2 [new Agent/TCP]
$ns attach-agent $n9 $tcp2
$tcp2 set fid_ 2
set tcpsink1 [new Agent/TCPSink]
$ns attach-agent $n11 $tcpsink1
$ns connect $tcp2 $tcpsink1

set cbr1 [new Application/Traffic/CBR]
$cbr1 set packet-size 200
$cbr1 set interval 0.005
$cbr1 attach-agent $udp1
$ns at 0.5 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"

set ftp0 [new Application/FTP]
$ftp0 set packet-size 200
$ftp0 set interval 0.005
$ftp0 attach-agent $tcp1
$ns at 0.5 "$ftp0 start"
$ns at 4.0 "$ftp0 stop"

set telnet0 [new Application/Telnet]
$telnet0 set packet-size 200
$telnet0 set interval 0.005
$telnet0 attach-agent $tcp2
$ns at 0.5 "$telnet0 start"
$ns at 4.5 "$telnet0 stop"

$ns at 5.0  "finish"
$ns run
```

Assignment 4

1 Parity check

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

void main()
{
```

```
char msg[20], s1[10] = {'1', '0'};
int i, op, count = 0;
printf("*** SENDER ***");
printf("\nEnter the message:");
scanf("%s", &msg);

for (i = 0; msg[i] != '\0'; i++)
{
    if (msg[i] == '1')
        count++;
}

int len = strlen(msg);
if (count % 2 == 0)
{
    sprintf(msg, "%s%c", msg, s1[0]);
    printf("Even Parity\n");
    printf("Message to be Transmitted:%s", msg);
}
else
{
    sprintf(msg, "%s%c", msg, s1[1]);
    printf("Odd Parity\n");
    printf("Message to be Transmitted:%s", msg);
}

printf("\nDo you want to introduce error(y/n):");
char ch;
int pos;
scanf(" %c", &ch);

if (ch == 'y')
{
x:
    printf("Enter the position:");
    scanf("%d", &pos);

    if (len < pos)
    {
        printf("\nInvalid position. Please re enter\n");
        goto x;
    }
    pos = pos - 1;

    if (msg[pos] == '1')
        msg[pos] = '0';
    else
```

```
            msg[pos] = '1';
    }

    printf("*** RECEIVER ***");
    printf("\nMessage received at the Receiver:%s", msg);
    len = strlen(msg);
    count = 0;

    for (i = 0; msg[i + 1] != '\0'; i++)
    {
        if (msg[i] == '1')
            count++;
    }

    if (msg[len - 1] == '0')
        if (count % 2 != 0)
            printf("\nMESSAGE WITHOUT ERROR");
        else
            printf("\nERROR IN MESSAGE");

    if (msg[len - 1] == '1')
        if (count % 2 == 0)
            printf("\nMESSAGE WITHOUT ERROR");
        else
            printf("\nERROR IN MESSAGE");
}
```

2 CRC

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

void crc(char message[], char pattern[])
{
    int msg_length = strlen(message);
    int pattern_length = strlen(pattern);
    int fcs_length = pattern_length - 1;
    char msg[20];
    strcpy(msg, message);

    // Performing CRC division
    int i;
    for (i = 0; i < msg_length; i++)
    {
        if (message[i] == '1')
        {
            if (i <= msg_length - pattern_length)
            {
                for (int j = 0; j < pattern_length; j++)
                {
                    if (message[i + j] == pattern[j])
                        message[i + j] = '0';
                    else
                        message[i + j] = '1';
                }
            }
            else
                break;
        }
    }

    printf("Remainder R is : %s\n", message);
    for (; i < msg_length + fcs_length; i++)
    {
        msg[i] = message[i];
    }

    printf("Message to be Transmitted T is : %s\n", msg);
    char choice;
    int position;
    printf("\nDo You Want to Introduce error(Y/N): ");
    scanf(" %c", &choice);

    if (choice == 'Y' || choice == 'y')
    {
    x:
        printf("Enter the Position: ");
        scanf("%d", &position);

        if (position < 1 || position > strlen(msg))
        {
            printf("Invalid position. Please enter a position between 1 and %d.\n",
strlen(msg));
            goto x;
        }

        // Introducing error at the specified position
        msg[msg_length - position] = (msg[msg_length - position] == '1') ? '0' :
'1';
    }

    printf("*** RECEIVER ***");
    printf("\nMessage received at the Receiver :%s\n", msg);
    for (i = 0; i < msg_length; i++)
    {
        if (msg[i] == '1')
        {
            if (i <= msg_length - pattern_length)
            {
                for (int j = 0; j < pattern_length; j++)
                {
                    if (msg[i + j] == pattern[j])
                        msg[i + j] = '0';
                    else
                        msg[i + j] = '1';
                }
            }
            else
                break;
        }
    }

    printf("Remainder R is : %s\n", msg);
    int flag = 0;

    for (i = 0; i < msg_length; i++)
    {
        if (msg[i] == '1')
        {
            flag = 1;
            break;
        }
    }

    if (flag)
    {
        printf("ERROR IN MESSAGE");
    }
    else
        printf("MESSAGE WITHOUT ERROR");
}

void main()
{
    char message[100];
    char pattern[100];
    char choice;
    int position;

    printf("*** SENDER ***");
    printf("\nEnter Message: ");
    scanf("%s", message);

    printf("Enter Pattern: ");
    scanf("%s", pattern);

    int msg_length = strlen(message);
    int pattern_length = strlen(pattern);

    if (pattern_length >= msg_length)
    {
        printf("Pattern size should be less than Message. Please re-enter the
Pattern \n");
        return 0;
    }

    printf("Given Message is  : %s\n", message);
    printf("Given Pattern is  : %s\n", pattern);
    printf("Message size is   : %d\n", msg_length);
    printf("Pattern Size is   : %d\n", pattern_length);
    printf("FCS               : %d bit\n", pattern_length - 1);

    // Appending zeros to the message
    for (int i = 0; i < pattern_length - 1; i++)
    {
        message[msg_length + i] = '0';
    }
    message[msg_length + pattern_length - 1] = '\0';

    printf("After Appending Q is: %s\n", message);
    crc(message, pattern);
}
```

<u>Assignment 5</u>
1 Simulate the transmission of **ping** messaged over a network topology consisting of 6 nodes.

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf

proc finish {} {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit(0)
}

set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]

$ns duplex-link $n1 $n5 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 5ms DropTail
$ns duplex-link $n2 $n4 1Mb 5ms DropTail
$ns duplex-link $n1 $n6 1Mb 5ms DropTail
$ns duplex-link $n3 $n6 1Mb 5ms DropTail

$ns queue-limit $n1 $n5 50
$ns queue-limit $n2 $n3 50
$ns queue-limit $n2 $n4 50
$ns queue-limit $n1 $n6 50
$ns queue-limit $n3 $n6 50

Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] has recieved ping answer $from with round trip time
$rtt ms"}

set p1 [new Agent/Ping]
$ns attach-agent $n1 $p1

set p2 [new Agent/Ping]
$ns attach-agent $n3 $p2

$ns connect $p1 $p2
$ns at 0.2 "$p1 send"
$ns at 0.6 "$p2 send"
$ns at 0.4 "$p1 send"
$ns at 0.8 "$p2 send"

$ns at 1.0 "finish"
$ns run
```

2 Write a Program to implement a R**ing** topology using less number of statements in TCL Language apply PING Agent

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf

proc finish {} {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit(0)
}

set numnodes 5

for { set i 0 } {$i<$numnodes} {incr i} {
set nodes($i) [$ns node]
}

for { set i 0 } {$i<$numnodes} {incr i} {
set j [expr {($i+1)%$numnodes}]
$ns duplex-link $nodes($i) $nodes($j) 1Mb 10ms DropTail }

Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] has received ping answer $from with round trip time of
$rtt ms"}

set p1 [new Agent/Ping]
$ns attach-agent $nodes(0) $p1

set p2 [new Agent/Ping]
$ns attach-agent $nodes(2) $p2

$ns connect $p1 $p2

$ns at 0.2 "$p1 send"
$ns at 0.6 "$p2 send"
$ns at 0.4 "$p1 send"
$ns at 0.8 "$p2 send"

$ns at 1.0 "finish"
$ns run
```

3 Write a Program to implement a **Star** topology using less number of statements in TCL Language using UDP and TCP Agent

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tf [open out.tr w]
$ns trace-all $tf

$ns color 1 red
$ns color 2 blue

proc finish {} {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam out.nam &
exit(0)
}

set numnodes 7

for { set i 0 } {$i<$numnodes} {incr i} {
set n($i) [$ns node]
}

for { set i 1 } {$i<7} {incr i} {
$ns duplex-link $n(0) $n($i) 1Mb 10ms DropTail }

$ns duplex-link-op $n(0) $n(1) orient left-up
$ns duplex-link-op $n(0) $n(2) orient right-up

$ns duplex-link-op $n(0) $n(3) orient left
$ns duplex-link-op $n(0) $n(4) orient left-down

$ns duplex-link-op $n(0) $n(5) orient right
$ns duplex-link-op $n(0) $n(6) orient right-down

set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
$udp0 set fid_ 1
set null1 [new Agent/Null]
$ns attach-agent $n(2) $null1
$ns connect $udp0 $null1
```

```
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packet-size 500
$cbr0 set interval 0.005
$cbr0 attach-agent $udp0
$ns at 0.3 "$cbr0 start"
$ns at 1.5 "$cbr0 stop"

set tcp0 [new Agent/TCP]
$ns attach-agent $n(3) $tcp0
$tcp0 set fid_ 2
set tcpsink0 [new Agent/TCPSink]
$ns attach-agent $n(5) $tcpsink0
$ns connect $tcp0 $tcpsink0

set ftp0 [new Application/FTP]
$ftp0 set packet-size 500
$ftp0 set interval 0.01
$ftp0 attach-agent $tcp0
$ns at 0.5 "$ftp0 start"
$ns at 2.0 "$ftp0 stop"

$ns at 2.5 "finish"
$ns run
```

Assignment 6

1 Implement hamming distance method for error correction. Input binary data and convert input data into code words in transmitted message. Display received message and display corrected message in case of error.

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

int pow1(int a, int b)
{
   int r = 1;

   for (int i = 0; i < b; i++)
   {
      r = r * a;
   }
   return r;
}

int check(char message[], int n)
{
   int len = strlen(message);
   int flag = 0, i = n - 1, count = 0;

   while (i < len)
   {
      int k = 0;

      while (k < n)
      {
          if (message[i++] == '1')
              count++;
          k++;
      }
      k = 0;

      while (k < n)
      {
          i++;
          k++;
      }
   }
   return count;
}

void main()
{
   char mess[20], nmess[50], hcode[10];
   int j = 0, k = 0, i = 0;

   printf("*** SENDER ***\n");
   printf("Enter Data:");
   scanf("%s", &mess);
   int len = strlen(mess);

   while (mess[i] != '\0')
   {
      if ((j + 1) == pow1(2, k))
      {
          nmess[j++] = ' ';
          nmess[j] = '\0';
          k++;
      }

      else
      {
          nmess[j++] = mess[i++];
          nmess[j] = '\0';
    }
}

int l = 0;
j = 0;

printf("\nAfter appending Empty space for message:%s\n", nmess);
len = strlen(nmess);

while (pow1(2, l) < len)
{
    int count = check(nmess, pow1(2, l));
    if (count % 2 == 0)
        hcode[j++] = '0';

    else
        hcode[j++] = '1';
    l++;
}

printf("\nCode Word:%s\n", hcode);
j = 0, i = 0, k = 0;

while (nmess[j] != '\0')
{
    if ((j + 1) == pow1(2, k))
    {
        nmess[j] = hcode[i++];
        k++;
    }
    j++;
}

int pos;
char ch;
printf("\nTransmission Data:%s", nmess);
printf("\nDo you want to introduce error (y/n):");
scanf(" %c", &ch);

if (ch == 'y')
{
    printf("\nEnter the position:");
x:
    scanf("%d", &pos);

    if (pos > len)
    {
            goto x;
        }

        else
        {
            nmess[pos - 1] = (nmess[pos - 1] == '1') ? '0' : '1';
        }
    }

    l = 0, j = 0;
    printf("*** RECEIVER ***\n");
    printf("Message received at the receiver:%s\n", nmess);

    while (pow1(2, l) < len)
    {
        int count = check(nmess, pow1(2, l));

        if (count % 2 == 0)
            hcode[j++] = '0';

        else
            hcode[j++] = '1';
        l++;
    }
    len = strlen(hcode);

    for (i = 0; i < len / 2; i++)
    {
        int temp = hcode[i];
        hcode[i] = hcode[len - i - 1];
        hcode[len - i - 1] = temp;
    }

    printf("Code Word:%s\n", hcode);
    int result = 0;
    pos = 0;

    for (i = len - 1; i >= 0; i--, pos++)
    {
        if (hcode[i] == '1')
            result += pow1(2, pos);
    }

    printf("Error in %d position\n", result);
    nmess[result - 1] = (nmess[result - 1] == '1') ? '0' : '1';
    printf("After Correction of Data:%s", nmess);
}
```

2 Write a program to implement Leaky Bucket algorithm.

```
#include <stdio.h>

struct ts
{
   int t;
   int ps;
} s[20];

void main()
{
   int n, outrate, time, np, i;

   printf("Enter the size of the bucket:");
   scanf("%d", &n);

   printf("Enter the outrate of the bucket:");
   scanf("%d", &outrate);

   printf("Enter the time interval of leak:");
   scanf("%d", &time);

   printf("Enter the number of packets:");
   scanf("%d", &np);

   for (i = 0; i < np; i++)
   {
      printf("Enter the time and size of packet%d: ", i);
      scanf("%d", &s[i].t);
      scanf("%d", &s[i].ps);
   }

   printf("oper  time  filled  free-space\n");
   int t[30], filled[30], free[30], k = 1, j = 0, flag;
   i = 0;

   do
   {
      flag = 0;
      if (i >= np)
      {
      l:
      {
         printf("remove  ");
```

```
    t[j] = time * k;
    filled[j] = filled[j - 1] - outrate;

    printf("%d      %d     ", t[j], filled[j]);
    k++;
}
}

// inserting
else if (s[i].t < time * k)
{
    printf("insert  ");
    t[j] = s[i].t;

    if (j == 0 && s[i].ps <= n)
    {
        filled[j] = s[i].ps;
        printf("%d      %d     ", t[j], filled[j]);
    }

    else if ((filled[j - 1] + s[i].ps) <= n)
    {
        filled[j] = filled[j - 1] + s[i].ps;
        printf("%d      %d     ", t[j], filled[j]);
    }

    else
    {
        flag = 1;
        printf("%d      overflow    ", t[j]);
    }
}

else
{
    i--;
    goto l;
}

if (flag == 0)
{
    free[j] = n - filled[j];
    printf("%d\n", free[j]);
}

else
{
```

```
            printf("\n");
            filled[j] = filled[j - 1];
        }

        j++;
        i++;
    } while (filled[j - 1] != 0);
}
```

Assignment 7

1 Distance

```
 #include <stdio.h>
#define INFINITY 9999
#define MAX 100

void Dijkstra(int cost[MAX][MAX], int n, int start, int destination)
{
    int distance[MAX], pred[MAX];
    int visited[MAX], count, mindistance, nextnode, i, j;

    // Creating cost matrix
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (cost[i][j] == 0)
                cost[i][j] = INFINITY;

            else
                cost[i][j] = cost[i][j];
        }
    }

    for (i = 0; i < n; i++)
    {
        distance[i] = cost[start][i];
        pred[i] = start;
        visited[i] = 0;
    }

    distance[start] = 0;
    visited[start] = 1;
    count = 1;

    while (count < n - 1)
    {
        mindistance = INFINITY;

        for (i = 0; i < n; i++)
        {
            if (distance[i] < mindistance && !visited[i])
            {
                mindistance = distance[i];
                nextnode = i;
            }
        }

        visited[nextnode] = 1;
        for (i = 0; i < n; i++)
        {
            if (!visited[i])
            {
                if (mindistance + cost[nextnode][i] < distance[i])
                {
                    distance[i] = mindistance + cost[nextnode][i];
                    pred[i] = nextnode;
                }
            }
        }
        count++;
    }

    // Printing the distance of each node from the source
    for (i = 0; i < n; i++)
    {
        if (i != start)
        {
            printf("Distance of Node %d: %d\n", i, distance[i]);
            printf("Route: %d", start);
            int path = pred[i];

            while (path != start)
            {
                printf(" --> %d ", path);
                path = pred[path];
            }
            printf(" --> %d\n", i);
        }
    }

    // Printing the shortest path
    printf("Shortest distance from source %d to destination %d: %d\n", start,
destination, distance[destination]);
```

```
    printf("Shortest path: %d ", start);
    int path = pred[destination];

    while (path != start)
    {
        printf(" --> %d ", path);
        path = pred[path];
    }
    printf(" --> %d\n", destination);
}

int main()
{
    int cost[MAX][MAX], i, j, n, u, destination;
    printf("Enter the number of nodes:");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (i > j)
            {
                printf("Distance between %d and %d: %d\n", i, j, cost[j][i]);
                cost[i][j] = cost[j][i];
            }

            else
            {
                printf("Distance between %d and %d:", i, j);
                scanf("%d", &cost[i][j]);
            }
        }
    }

    printf("Enter the starting node:");
    scanf("%d", &u);

    printf("Enter the destination node:");
    scanf("%d", &destination);

    Dijkstra(cost, n, u, destination);
    return 0;
}
```

Assignment 8

1
```
#include <stdio.h>

struct node
{
   unsigned dist[20];
   unsigned from[20];
} rt[10];

int main()
{
   int dmat[20][20];
   int n, i, j, k, count = 0;

   printf("Enter the number of nodes: ");
   scanf("%d", &n);

   printf("\nEnter the cost matrix:(Assign value 999 if there is no direct
connection)\n");
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
      {
         scanf("%d", &dmat[i][j]);
         dmat[i][i] = 0;
         rt[i].dist[j] = dmat[i][j];
         rt[i].from[j] = j;
      }

   printf("\nInitial Tables:\n");
   for (i = 0; i < n; i++)
   {
      printf("\nRouter %d:\n", i + 1);
      printf("Destination\tDistance\tNext Hop\n");

      for (j = 0; j < n; j++)
      {
         if (rt[i].dist[j] == 999)
         {
            printf("%d\t\t%d\t\t-\n", j + 1, rt[i].dist[j]);
         }
         else
         {
            printf("%d\t\t%d\t\t%d\n", j + 1, rt[i].dist[j], rt[i].from[j] + 1);
         }
      }
   }
```

```
    do
    {
        count = 0;
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                for (k = 0; k < n; k++)

                    if (rt[i].dist[j] > dmat[i][k] + rt[k].dist[j])
                    {
                        rt[i].dist[j] = rt[i].dist[k] + rt[k].dist[j];
                        rt[i].from[j] = k;
                        count++;
                    }
    }

    while (count != 0);
    printf("\nUpdated Tables:\n");

    for (i = 0; i < n; i++)
    {
        printf("\nRouter %d:\n", i + 1);
        printf("Destination\tDistance\tNext Hop\n");

        for (j = 0; j < n; j++)
        {
            printf("%d\t\t%d\t\t%d\n", j + 1, rt[i].dist[j], rt[i].from[j] + 1);
        }
    }

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

Assignment 9

1

```
//Server.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

int main(){
```

```
/*
Define the IP (Internet Protocol) address and port number, that would be used
to create a
socket. Here, we are using the localhost address for both the server and the
client.
*/
char *ip = "127.0.0.1";
int port = 5566;

//define the variables

int server_sock, client_sock;
struct sockaddr_in server_addr, client_addr;
socklen_t addr_size;
char buffer[1024];
int n;server_sock = socket(AF_INET, SOCK_STREAM, 0);
if (server_sock < 0){
perror("[-]Socket error");
exit(1);
}
printf("[+]TCP server socket created.\n");

/*
Here, we initialize the server address by providing the required
IP and port number. The server keeps all the address information for
both the server and the client in the sockaddr_in struct.
*/

memset(&server_addr, '\0', sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = port;
server_addr.sin_addr.s_addr = inet_addr(ip);

//Binding the socket descriptor with the server address

//information.

n = bind(server_sock, (struct sockaddr*)&server_addr,sizeof(server_addr));
if (n < 0){
perror("[-]Bind error");
exit(1);
}
printf("[+]Bind to the port number: %d\n", port);

//Now, we listen for incoming connections from the clients.

listen(server_sock, 5);
```

```
printf("Listening...\n");

/*
the server handles only one client at a time. So, only one client
would communicate and the rest would have to wait for the
communication to get completed.
*/

char c[200];

while(1){
addr_size = sizeof(client_addr);
client_sock = accept(server_sock, (struct sockaddr*)&client_addr,
&addr_size);
printf("[+]Client connected.\n");
while(1){
bzero(buffer, 1024);
recv(client_sock, buffer, sizeof(buffer), 0);
printf("Client: %s\n", buffer);

bzero(buffer, 1024);

printf("Enter the message : ");

fgets(c,200,stdin);

strcpy(buffer, c);
printf("Server: %s\n", buffer);
send(client_sock, buffer, strlen(buffer), 0);

}

close(client_sock);
printf("[+]Client disconnected.\n\n");


}

}


// Client.c

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

```
#include <arpa/inet.h>

int main()
{
   /*
   NOTE: Make sure that the IP address and the port number for the client are
the same as
   the server.
   */
   char *ip = "127.0.0.1";
   int port = 5566;
   int sock;
   struct sockaddr_in addr;
   socklen_t addr_size;
   char buffer[1024];
   int n;

   /*
   We start by creating a TCP (Transmission Control Protocol) socket. The
socket would be
   used to connect to the server and start communication.
   */

   sock = socket(AF_INET, SOCK_STREAM, 0);
   if (sock < 0)
   {
      perror("[-]Socket error");
      exit(1);
   }
   printf("[+]TCP server socket created.\n");

   // provide the required IP address and the port number to the required data
structures.
   memset(&addr, '\0', sizeof(addr));
   addr.sin_family = AF_INET;
   addr.sin_port = port;
   addr.sin_addr.s_addr = inet_addr(ip);

   // We send a connection request to the server and wait for the server to
accept the request.

   connect(sock, (struct sockaddr *)&addr, sizeof(addr));
   printf("Connected to the server.\n");

   // We send a message to the server and wait for the reply.
   bzero(buffer, 1024);
```

```
    char k[200];
    while (1)
    {

        printf("Enter the message : ");

        fgets(k, 200, stdin);
        strcpy(buffer, k);

        printf("Client: %s\n", buffer);
        send(sock, buffer, strlen(buffer), 0);

        /* We receive the reply from the server and print it on the console.*/

        bzero(buffer, 1024);
        recv(sock, buffer, sizeof(buffer), 0);
        printf("Server: %s\n", buffer);
    }

    // At last, we close the connection from the server.
    close(sock);
    printf("Disconnected from the server.\n");
    return 0;
}


//Arithmetic operations server client in java

### Server Code

import java.io.*;
import java.net.*;

public class ArithmeticServer {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        Socket clientSocket = null;

        try {
            serverSocket = new ServerSocket(9876);
            System.out.println("Server is listening on port 9876...");

            while (true) {
                clientSocket = serverSocket.accept();
                System.out.println("Client connected!");
                DataInputStream input = new
DataInputStream(clientSocket.getInputStream());
                DataOutputStream output = new
DataOutputStream(clientSocket.getOutputStream());

                double number1 = input.readDouble();
                double number2 = input.readDouble();
                char operator = input.readChar();

                double result = performOperation(number1, number2, operator);
                output.writeDouble(result);

                System.out.println("Operation performed: " + number1 + " " +
operator + " " + number2 + " = " + result);

                input.close();
                output.close();
                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (serverSocket != null) serverSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private static double performOperation(double number1, double number2,
char operator) {
        switch (operator) {
            case '+':
                return number1 + number2;
            case '-':
                return number1 - number2;
            case '*':
                return number1 * number2;
            case '/':
                if (number2 != 0) {
                    return number1 / number2;
                } else {
                    System.out.println("Error: Division by zero");
                    return Double.NaN;
                }
            default:
                System.out.println("Error: Invalid operator");
                return Double.NaN;
        }
    }
}
```

Client Code

```java
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class ArithmeticClient {
    public static void main(String[] args) {
        Socket socket = null;
        DataInputStream input = null;
        DataOutputStream output = null;
        Scanner scanner = new Scanner(System.in);

        try {
            socket = new Socket("localhost", 9876);
            System.out.println("Connected to server");

            input = new DataInputStream(socket.getInputStream());
            output = new DataOutputStream(socket.getOutputStream());

            System.out.print("Enter first number: ");
            double number1 = scanner.nextDouble();

            System.out.print("Enter second number: ");
            double number2 = scanner.nextDouble();

            System.out.print("Enter an operator (+, -, *, /): ");
            char operator = scanner.next().charAt(0);

            output.writeDouble(number1);
            output.writeDouble(number2);
            output.writeChar(operator);

            double result = input.readDouble();
            System.out.println("Result: " + result);

        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (scanner != null) scanner.close();
                if (input != null) input.close();
                if (output != null) output.close();
                if (socket != null) socket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Explanation

1. **Server Code**:
   - The server listens on port 9876.
   - When a client connects, the server reads two numbers and an arithmetic operator from the client.
   - It performs the requested arithmetic operation using a helper method `performOperation`.
   - The result is sent back to the client.
   - The server handles each client in a sequential manner.

2. **Client Code**:
   - The client connects to the server at `localhost` on port 9876.
   - It prompts the user to enter two numbers and an arithmetic operator.
   - These inputs are sent to the server.
   - The client reads the result from the server and displays it.