

PART-B**Introduction to JAVA**

Java is a general-purpose computer programming language that is simple, concurrent, class-based, object-oriented language. The compiled Java code can run on all platforms that support Java without the need for recompilation hence Java is called as "write once, run anywhere" (WORA). The Java compiled intermediate output called "byte-code" that can run on any Java virtual machine (JVM) regardless of computer architecture. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

Java is defined by a specification and consists of a programming language, a compiler, core libraries and a runtime (Java virtual machine). The Java runtime allows software developers to write program code in other languages than the Java programming language which still runs on the Java virtual machine. The *Java platform* is usually associated with the *Java virtual machine* and the *Java core libraries*.

In Linux operating system Java libraries are preinstalled. It's very easy and convenient to compile and run Java programs in Linux environment. To compile and run Java Program is a two-step process:

1. Compile Java Program from Command Prompt

```
[root@host ~]# javac MyProgram.java
```

The Java compiler (Javac) compiles java program and generates a byte-code with the same file name and .class extension.

2. Run Java program from Command Prompt

```
[root@host ~]# java MyProgram
```

The java interpreter (Java) runs the byte-code and gives the respective output. It is important to note that in above command we have omitted the .class suffix of the byte-code (Filename.class).



Experiment No: 1

Error Detecting Code Using CRC-CCITT (16-bit)

Objective : Write a program for error detecting code using CRC-CCITT (16- bits).

Theory :

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The **CRC** calculation or *cyclic redundancy check* was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used.

The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation— by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar.

$$\begin{array}{r}
 101 = 5 \\
 ----- \\
 10011 \, / \, 1101101 \\
 10011 \, | \, | \\
 ----- \, | \, | \\
 10000 \, | \\
 00000 \, | \\
 ----- \, | \\
 10000 \, 1 \\
 10011 \\
 ----- \\
 1110 = 14 = \text{remainder}
 \end{array}$$

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value "Lammert"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an *exclusive or* operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

- The message bits are appended with c zero bits; this *augmented message* is the dividend
- A predetermined $c+1$ -bit binary sequence, called the *generator polynomial*, is the divisor
- The checksum is the c -bit remainder that results from the division operation

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

	CRC-CCITT	CRC-16	CRC-32
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	10000010011000001000111011011011

International Standard CRC Polynomials

Algorithm :

Input: Frame data.

Output: Data with or without errors.

Step 1. Start

Step 2. Given a bit string, append $(n-1)$ 0's at the end where n is the size of some agreed on Polynomial $G(x)$.

Step 3. Divide $B(x)$ by polynomial $G(x)$ and determine the remainder $R(x)$.

Step 4. Define $T(x) = B(x) - R(x)$

$$(T(x)/G(x) \Rightarrow \text{remainder } 0)$$

Step 5. Transmit T , the bit string corresponding to $T(x)$.

Step 6: Let T' represent the bit stream the receiver gets and $T'(x)$ the associated polynomial. The receiver divides $T'(x)$ by $G(x)$. If there is a 0 remainder, the receiver concludes $T = T'$ and no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.

Step 7: stop

Program:

```
import java.io.*;
class Crc
{
    public static void main(String args[]) throws
    IOException {
        BufferedReader br=new BufferedReader(new
        InputStreamReader(System.in));
        int[ ] data;
        int[ ]div;
        int[ ]divisor;
        int[ ]rem;
        int[ ] crc;
        int data_bits, divisor_bits, tot_length;
        System.out.println("Enter number of data bits : ");
        data_bits=Integer.parseInt(br.readLine());
        data=new int[data_bits];
        System.out.println("Enter data bits : ");
        for(int i=0; i<data_bits; i++)
            data[i]=Integer.parseInt(br.readLine());
        System.out.println("Enter number of bits in
        divisor : ");
        divisor_bits=Integer.parseInt(br.readLine());
        divisor=new int[divisor_bits];
        System.out.println("Enter Divisor bits : ");
        for(int i=0; i<divisor_bits; i++)
            divisor[i]=Integer.parseInt(br.readLine());
        System.out.print("Data bits are : ");
        for(int i=0; i< data_bits; i++)
            System.out.print(data[i]);
        System.out.println();
        System.out.print("divisor bits are : ");
        for(int i=0; i< divisor_bits; i++)
            System.out.print(divisor[i]);
        System.out.println();
        tot_length=data_bits+divisor_bits-1;
        div=new int[tot_length];
        rem=new int[tot_length];
```

```
crc=new int[tot_length];

/*----- CRC GENERATION-----*/

for(int i=0;i<data.length;i++)
    div[i]=data[i];

System.out.print("Dividend (after appending 0's) are : ");
for(int i=0; i< div.length; i++) System.out.print(div[i]);

System.out.println();

for(int j=0; j<div.length; j++){
    rem[j] = div[j];
}

rem=divide(div, divisor, rem);
for(int i=0;i<div.length;i++)    //append dividend and remainder
{
    crc[i]=(div[i]^rem[i]);
}

System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
    System.out.print(crc[i]);

/*-----ERROR DETECTION-----*/

System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
    crc[i]=Integer.parseInt(br.readLine());

System.out.print("crc bits are : ");
for(int i=0; i< crc.length; i++)
    System.out.print(crc[i]);

System.out.println();
for(int j=0; j<crc.length; j++){
    rem[j] = crc[j];
}

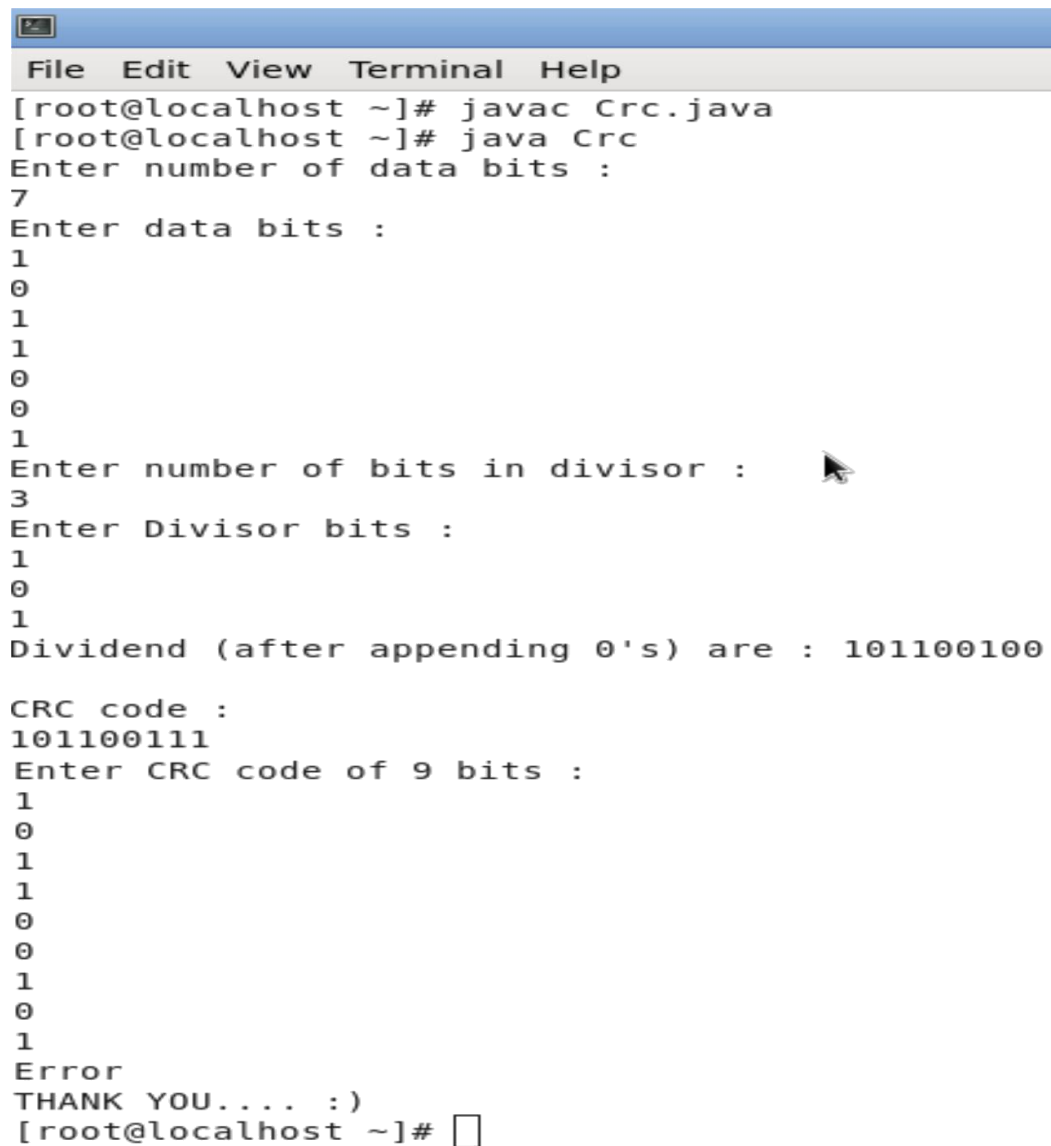
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
    if(rem[i]!=0)
```

```
        {
            System.out.println("Error");
            break;
        }
        if(i==rem.length-1)
            System.out.println("No Error");
    }
    System.out.println("THANK YOU.... :)");
}

static int[] divide(int div[],int divisor[], int rem[])
{
    int cur=0;
    while(true)
    {
        for(int i=0;i<divisor.length;i++)
            rem[cur+i]=(rem[cur+i]^divisor[i]);
        while(rem[cur]==0 && cur!=rem.length-1)
            cur++;
        if((rem.length-cur)<divisor.length))
            break;
    }
    return rem;
}
}
```

Output:

[root@localhost ~]# gedit Crc.java



```
File Edit View Terminal Help
[root@localhost ~]# javac Crc.java
[root@localhost ~]# java Crc
Enter number of data bits :
7
Enter data bits :
1
0
1
1
0
0
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 101100100

CRC code :
101100111
Enter CRC code of 9 bits :
1
0
1
1
0
0
1
0
1
Error
THANK YOU.... :)
[root@localhost ~]#
```

Experiment No: 2

Bellman-Ford Algorithm

Objective : Write a program to find the shortest path between vertices using bellman-ford algorithm

Theory :

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbors of its routing table. For each network path, the receiving routers pick the neighbor advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: the Bellman-Ford and the Dijkstra algorithms.

Routers that use this algorithm have to maintain the distance tables (which is a one-dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up date by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbors whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc.

The Bellman-Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman-Ford algorithm can detect negative cycles and report their existence.

Algorithm:

```
function BellmanFord(list vertices, list edges, vertex source)
    ::distance[],predecessor[]
// Step 1: initialize graph
for each vertex v in vertices:
    distance[v] := inf
    predecessor[v] := null
distance[source] := 0
// Step 2: relax edges repeatedly
for i from 1 to size(vertices)-1:
    for each edge (u, v) with weight w in edges:
        if distance[u] + w < distance[v]:
            distance[v] := distance[u] + w
            predecessor[v] := u
// Step 3: check for negative-weight cycles
for each edge (u, v) with weight w in edges:
    if distance[u] + w < distance[v]:
        error "Graph contains a negative-weight cycle"
return distance[], predecessor[]
```


Program:

```
import java.util.Scanner;

public class BellmanFord
{
    private int D[];
    private int num_ver;
    public static final int MAX_VALUE = 999;
    public BellmanFord(int num_ver)
    {
        this.num_ver = num_ver;
        D = new int[num_ver + 1];
    }

    public void BellmanFordEvaluation(int source, int A[][])
    {
        for (int node = 1; node <= num_ver; node++)
        {
            D[node] = MAX_VALUE;
        }
        D[source] = 0;
        for (int node = 1; node <= num_ver - 1; node++)
        {
            for (int sn = 1; sn <= num_ver; sn++)
            {
                for (int dn = 1; dn <= num_ver; dn++)
                {
                    if (A[sn][dn] != MAX_VALUE)
                    {
                        if (D[dn] > D[sn] + A[sn][dn])
                            D[dn] = D[sn] + A[sn][dn];
                    }
                }
            }
        }
        for (int sn = 1; sn <= num_ver; sn++)
        {
            for (int dn = 1; dn <= num_ver; dn++)
```

```
        {
            if (A[sn][dn] != MAX_VALUE)
            {
                if (D[dn] > D[sn] + A[sn][dn])

                    System.out.println("The Graph contains negative egde
                    cycle");
            }
        }
    }
    for (int vertex = 1; vertex <= num_ver; vertex++)
    {
        System.out.println("distance of source " + source + " to " + vertex + "
        is " + D[vertex]);
    }
}

public static void main(String[ ] args)
{
    int num_ver = 0;
    int source;
    Scanner scanner = new
    Scanner(System.in);
    System.out.println("Enter the number of
    vertices"); num_ver = scanner.nextInt();
    int A[][] = new int[num_ver + 1][num_ver + 1];
    System.out.println("Enter the adjacency matrix");
    for (int sn = 1; sn <= num_ver; sn++)
    {
        for (int dn = 1; dn <= num_ver; dn++)
        {

            A[sn][dn] = scanner.nextInt();
            if (sn == dn)
            {
                A[sn][dn] = 0;
                continue;
            }
            if (A[sn][dn] == 0)
```

```

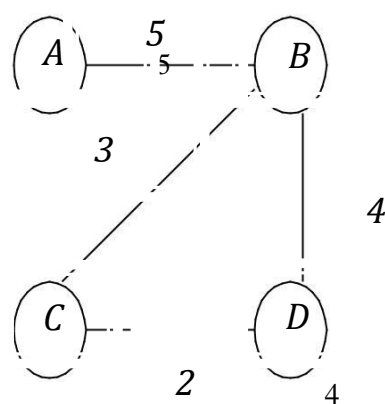
        {
            A[sn][dn] = MAX_VALUE;
        }
    }

    }

    System.out.println("Enter the source vertex");
    source = scanner.nextInt();

    BellmanFord b = new BellmanFord
    (num_ver); b.BellmanFordEvaluation(source,
    A); scanner.close();
}
}

```



Output:

```
[root@localhost ~]# gedit BellmanFord.java
```

```

root@localhost:~
File Edit View Terminal Help
[root@localhost ~]# javac BellmanFord.java
[root@localhost ~]# java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix
0 5 0 0
5 0 3 4
0 3 0 2
0 4 2 0
Enter the source vertex
2
distance of source 2 to 1 is 5
distance of source 2 to 2 is 0
distance of source 2 to 3 is 3
distance of source 2 to 4 is 4
[root@localhost ~]# 

```

Experiment No 3:

Program to find the IP address of the given hostname:

Objective:

The objective of this program is to develop a Java-based application that takes a hostname as input and resolves its corresponding IP address using the `InetAddress` class. The program will print the IP address of the given hostname or an error message if the hostname cannot be resolved.

Theory:

The Domain Name System (DNS) is a system used to map human-readable domain names, like `www.google.com`, to IP addresses, which are numeric identifiers used for routing data over networks. Every time you visit a website, your browser uses DNS to translate the domain name into its corresponding IP address so that the network can route the traffic to the correct destination.

In Java, the `InetAddress` class, part of the `java.net` package, provides methods to retrieve the IP address for a given hostname. The `InetAddress.getByName()` method performs the DNS resolution, returning an `InetAddress` object. The `getHostAddress()` method of `InetAddress` can then be used to get the textual representation of the IP address.

Key Concepts:

1. **DNS Lookup:** Resolving a hostname to an IP address.
2. **InetAddress Class:** Provides methods to interact with IP addresses and domain names.
3. **IP Address:** The numeric representation of a device's network address.

Algorithm:

1. **Start.**
2. **Input:** Accept the hostname (e.g., `www.example.com`) from the user.
3. **Process:**
 1. Use the `InetAddress.getByName(hostname)` method to get the IP address of the given hostname.
 2. If the hostname is valid, extract the IP address using `InetAddress.getHostAddress()`.
4. **Output:** Print the IP address of the hostname.
5. **Error Handling:**
 1. If the hostname cannot be resolved, catch the `UnknownHostException` and print an error message indicating that the hostname could not be resolved.
6. **End.**

```
import java.io.*;
import java.util.*;
import java.net.*;
class GFG {
```

```
public static void main(String[] args)
{
    // Host
    String host = "www.geeksforgeeks.org"; // we can change this like www.google.com
    etc..

    // Using try Logic So that if there is an error then
    // easily get the error
    try {
        // calling the function which gives the IP
        // Address from the given host
        InetAddress[] iaddress = InetAddress.getAllByName(host);

        for (InetAddress ipaddresses : iaddress)
        {
            System.out.println(ipaddresses.toString());
        }
    }
    catch (UnknownHostException e) {
        System.out.println(e);
    }
}
```

OUTPUT:

Enter the hostname: www.google.com

IP address of www.google.com is: 142.250.190.196

Experiment No: 4

Program to determine the IP address and host name of the local computer.

Objective:

The objective of this program is to determine and display the **IP address** and **hostname** of the local computer using Java. The program will retrieve the IP address and hostname of the machine on which it is executed using the `InetAddress` class.

Theory:

Every device connected to a network has a unique identifier known as an **IP (Internet Protocol) address**. This address is used to route data to the correct location on the internet or within a network. Each device also has a **hostname**, which is the name assigned to it on the network.

In Java, the `InetAddress` class from the `java.net` package provides methods to retrieve the IP address and hostname of the local machine. The method `InetAddress.getLocalHost()` returns the `InetAddress` object for the local machine. From this object, we can retrieve:

1. **Hostname:** Using the `getHostName()` method.
 2. **IP Address:** Using the `getHostAddress()` method.
-

Algorithm:

1. **Start.**
2. **Retrieve Local Host Information:**
 1. Use `InetAddress.getLocalHost()` to get the local host's `InetAddress` object.
 2. From this object, use `getHostName()` to get the local hostname.
 3. Use `getHostAddress()` to get the IP address of the local machine.
3. **Display** the local hostname and IP address on the console.
4. **Error Handling:**
 - If an error occurs, such as if the local host name cannot be determined, catch the `UnknownHostException` and display an error message.
5. **End.**

Source Code:

```
import java.io.*;
import java.net.InetAddress;

public class GFG {
    public static void main(String[] args) throws Exception
    {
        // a variable of type InetAddress to store
```

```
// the address of the local host
InetAddress addr = InetAddress.getLocalHost();
// Returns the IP address string in
// textual presentation.
System.out.println("Local HostAddress: "
    + addr.getHostAddress());
// Gets the host name for this IP address.
System.out.println("Local host name: "
    + addr.getHostName());
}
}
```

Output

Local HostAddress: 127.0.0.1

Local host name: localhost

Experiment No: 5**Client –Server Using TCP/IP Sockets**

Objective : Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present. Implement the above program using as message queues or FIFOs as IPC channels.

Theory :

Socket is an interface which enables the client and the server to communicate and pass on information from one another. Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

Algorithm :**Server Side :**

1. Create socket,port =x for incoming request
serverSocket=socket()
2. Wait for incoming connection request
connectionSocket =serverSocket.accept()
3. Read request from **connectionSocket**
4. Write reply to **connectionSocket**
5. Close **connectionSocket**

Client Side :

1. create socket,connect to **hostid**, port=x
clientSocket = socket()
2. send request using **clientSocket**
3. read reply from **clientSocket**
4. close **clientSocket**

Program:

TCP Client:

```
import java.net.*;
import java.io.*;
public class TCPClient
{
    public static void main(String args[]) throws Exception
    {
        Socket clientsock=new Socket("127.0.0.1",4000);
        System.out.println("Connection Successfull");
        System.out.println("Enter the File Name\n");
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        String fname = br.readLine();
        OutputStream ostream =clientsock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream,true);
        pwrite.println(fname);
        InputStream istream = clientsock.getInputStream();
        BufferedReader socketRead = new BufferedReader(new
        InputStreamReader(istream));
        String str;
        while((str=socketRead.readLine())!=null)
        {
            System.out.println(str);
        }
    }
}
```

TCPServer program

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
public class TCPServer
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket sersock = new ServerSocket(4000);
        System.out.println("Sever ready for Connection");
        Socket sock = sersock.accept();
        System.out.println("Waiting for filename");
        InputStream istream = sock.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(istream));
        String fname = br.readLine();
        OutputStream ostream = sock.getOutputStream();
    }
}
```

```
PrintWriter pwrite = new PrintWriter(ostream,true);
String str;
try{
BufferedReader contentRead = new BufferedReader(new FileReader(fname));
while((str= contentRead.readLine())!=null)
{
pwrite.println(str);
}
System.out.println("File Contents sent Successfully");
}
catch (FileNotFoundException e){
pwrite.println("File Not Found at the Server");
System.out.println("File Not Found");
}
}
}
```

Note: Create two different files Client.java and Server.java. Follow the steps given:

1. Open a terminal run the server program and provide the filename to send
2. Open one more terminal run the client program and provide the IP address of the server. We can give localhost address "127.0.0.1" as it is running on same machine or give the IP address of the machine.
3. Send any start bit to start sending file.

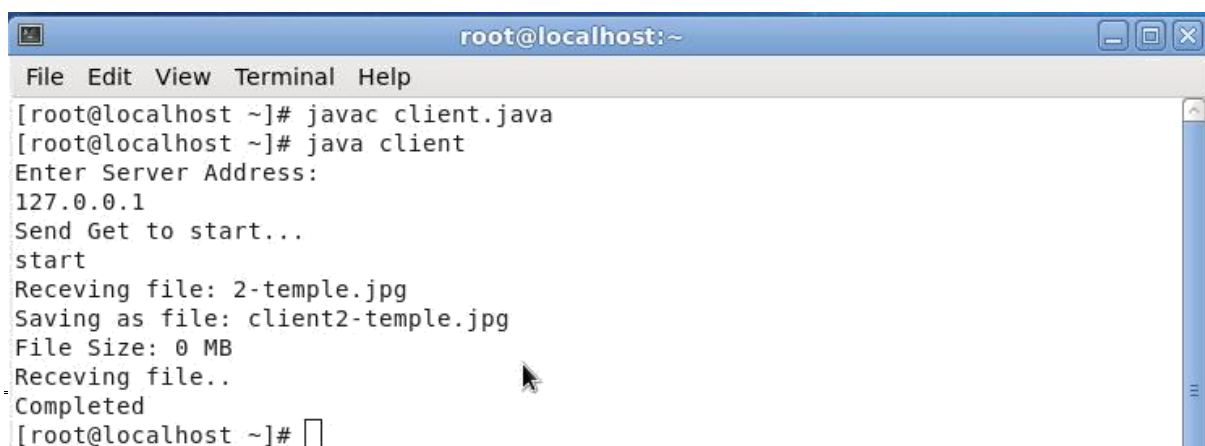
Output:

At server side:



```
root@localhost:~
File Edit View Terminal Help
[root@localhost ~]# javac server.java
[root@localhost ~]# java server
Enter File Name:
2-temple.jpg
Waiting for request
Connected With /127.0.0.1
SendGet....Ok
Sending File: 2-temple.jpg
Size: 19868
Buf size: 43690
..ok
Send Complete
Waiting for request
□
```

At client side:



```
root@localhost:~
File Edit View Terminal Help
[root@localhost ~]# javac client.java
[root@localhost ~]# java client
Enter Server Address:
127.0.0.1
Send Get to start...
start
Receiving file: 2-temple.jpg
Saving as file: client2-temple.jpg
File Size: 0 MB
Receiving file..
Completed
[root@localhost ~]# □
```

Experiment No: 6**Client –Server Communication using UDP**

Objective : Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.

Theory :

A datagram socket is the one for sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

Algorithm :**Server Side :**

1. Create socket, port= x
serverSocket =socket(AF_INET,SOCK_DGRAM)
2. read datagram from **serverSocket**
3. write reply to **serverSocket** specifying client address,port number

Client Side :

1. create socket:
clientSocket =socket(AF_INET,SOCK_DGRAM)
2. Create datagram with server IP and port=x; send datagram **via clientSocket**
3. read datagram from **clientSocket**
4. close **clientSocket**

Program :

```
import java.io.*;
import java.net.*;
class UDPC
{
public static void main(String args[]) throws Exception
{
DatagramSocket clientsocket=new DatagramSocket();
BufferedReader clientread=new BufferedReader(new
InputStreamReader(System.in));
InetAddress ip=InetAddress.getByName("127.0.0.1");
byte[] sendbuffer=new byte[1024];
byte[] receivebuffer=new byte[1024];
```

```
System.out.println("Client\n");
String clientdata=clientread.readLine();
sendbuffer=clientdata.getBytes();
DatagramPacket sendpacket=new
DatagramPacket(sendbuffer,sendbuffer.length,ip,9876);
clientsocket.send(sendpacket);
DatagramPacket receivepacket=new
DatagramPacket(receivebuffer,receivebuffer.length);
clientsocket.receive(receivepacket);
String serverdata=new String(receivepacket.getData());
System.out.println("\nserver msg"+serverdata);
clientsocket.close();
}
}
```

UDPS Server

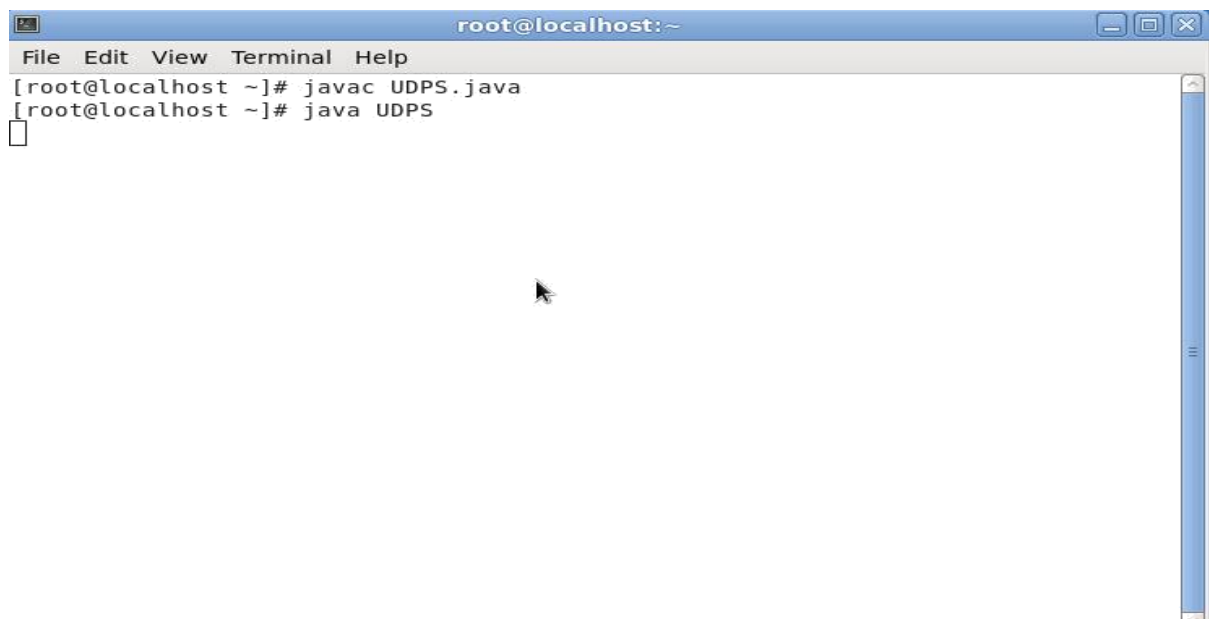
```
import java.io.*;
import java.net.*;
class UDPS
{
public static void main(String args[]) throws Exception
{
DatagramSocket serversocket=new DatagramSocket(9876);
byte[] sendbuffer=new byte[1024];
byte[] receivebuffer=new byte[1024];
while(true)
{
DatagramPacket receivepacket=new
DatagramPacket(receivebuffer,receivebuffer.length);
serversocket.receive(receivepacket);
InetAddress ip=receivepacket.getAddress();
int portno=receivepacket.getPort();
String clientdata=new String(receivepacket.getData());
System.out.println("clientmsg"+clientdata);
System.out.println("Server\n");
BufferedReader serverread=new BufferedReader(new
InputStreamReader(System.in));
String serverdata=serverread.readLine();
sendbuffer=serverdata.getBytes();
DatagramPacket sendpacket=new
DatagramPacket(sendbuffer,sendbuffer.length,ip,portno);
```

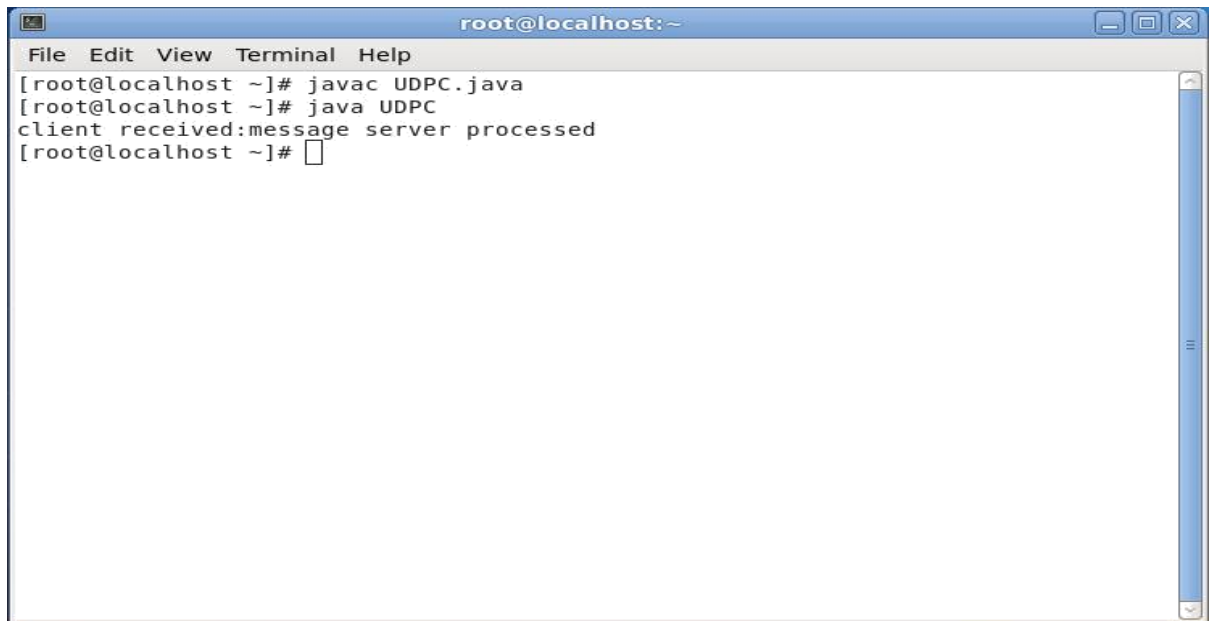
```
serversocket.send(sendpacket);  
}  
}  
}
```

Note: Create two different files UDPC.java and UDPS.java. Follow the following steps:

1. Open a terminal run the server program.
2. Open one more terminal run the client program, the sent message will be received.

At Server side:



At Client side:

A terminal window titled "root@localhost:~" with a menu bar (File, Edit, View, Terminal, Help). The terminal shows the following commands and output:

```
[root@localhost ~]# javac UDPC.java
[root@localhost ~]# java UDPC
client received:message server processed
[root@localhost ~]#
```

Experiment No: 7

RSA Algorithm to Encrypt and Decrypt the Data

Objective : Write a program for simple RSA algorithm to encrypt and decrypt the data.

Theory

RSA is an example of public key cryptography. It was developed by Rivest, Shamir and Adelman. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

The RSA algorithm's efficiency requires a fast method for performing the modular exponentiation operation. A less efficient, conventional method includes raising a number (the input) to a power (the secret or public key of the algorithm, denoted e and d , respectively) and taking the remainder of the division with N . A straight-forward implementation performs these two steps of the operation sequentially: first, raise it to the power and second, apply modulo. The RSA algorithm comprises of three steps, which are depicted below:

Algorithm

Key Generation Algorithm

1. Generate two large random primes, p and q , of approximately equal size such that their product $n = p \cdot q$
2. Compute $n = p \cdot q$ and Euler's totient function (ϕ) $\phi(n) = (p-1)(q-1)$.
3. Choose an integer e , $1 < e < \phi$, such that $\gcd(e, \phi) = 1$.
4. Compute the secret exponent d , $1 < d < \phi$, such that $e \cdot d \equiv 1 \pmod{\phi}$.
5. The public key is (e, n) and the private key is (d, n) . The values of p , q , and ϕ should also be kept secret.

Encryption

Sender A does the following:-

1. Using the public key (e, n)
2. Represents the plaintext message as a positive integer M
3. Computes the cipher text $C = M^e \pmod{n}$.
4. Sends the cipher text C to B (Receiver).

Decryption

Recipient B does the following:-

1. Uses his private key (d, n) to compute $M = C^d \bmod n$.
2. Extracts the plaintext from the integer representative m.

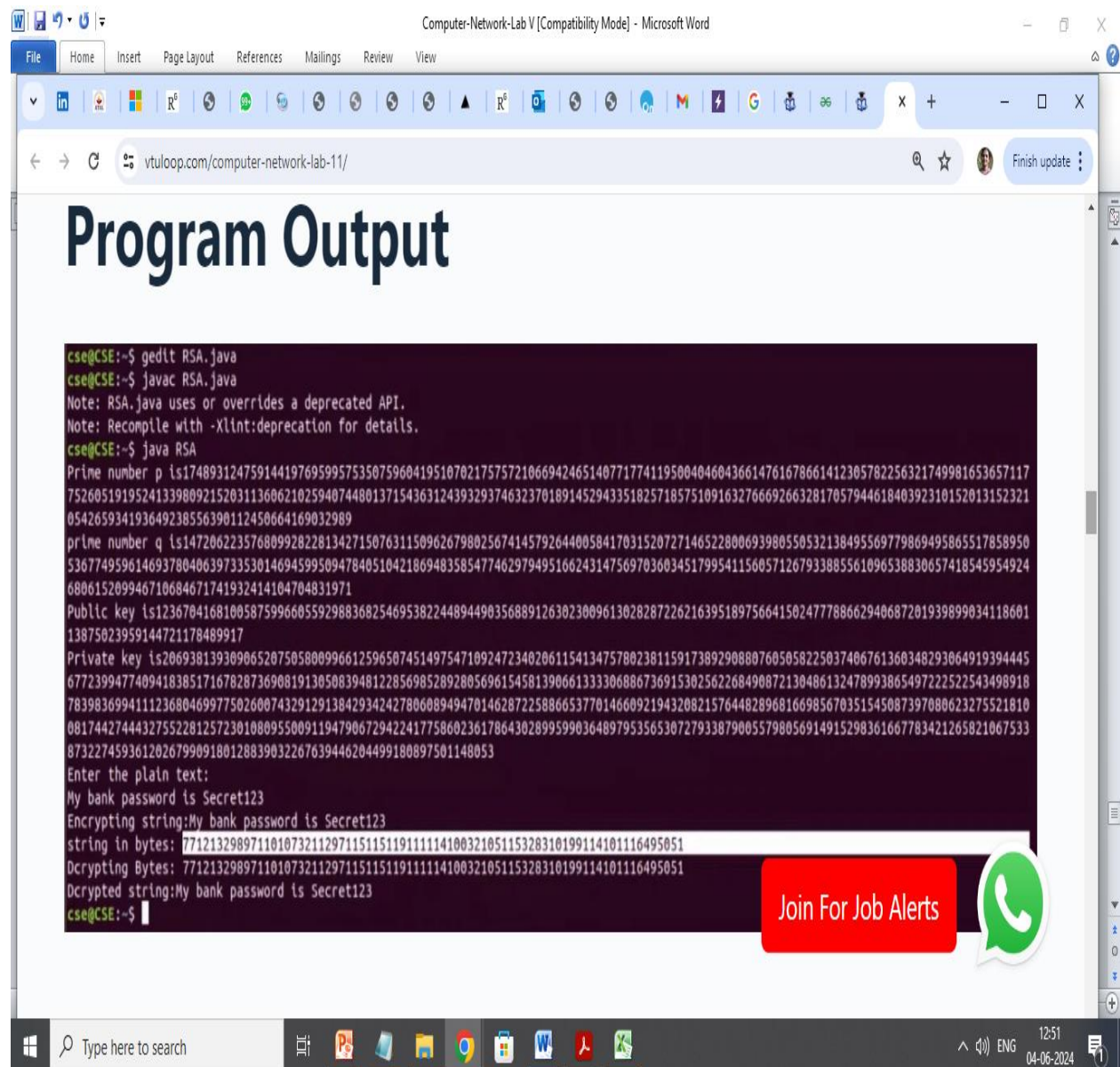
Program

```
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;

public class RSA
{
    private BigInteger p,q,N,phi,e,d;
    private int bitlength=1024;
    private Random r;
    public RSA()
    {
        r=new Random();
        p=BigInteger.probablePrime(bitlength,r);
        q=BigInteger.probablePrime(bitlength,r);
        System.out.println("Prime number p is"+p);
        System.out.println("prime number q is"+q);
        N=p.multiply(q);
        phi=p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e=BigInteger.probablePrime(bitlength/2,r);
        while(phi.gcd(e).compareTo(BigInteger.ONE)>0&&e.compareTo(phi)<0)
        {
            e.add(BigInteger.ONE);
        }
        System.out.println("Public key is"+e);
        d=e.modInverse(phi);
        System.out.println("Private key is"+d);
    }
    public RSA(BigInteger e,BigInteger d,BigInteger N)
    {
        this.e=e;
```



```
this.d=d;
this.N=N;
}
public static void main(String[] args)throws IOException
{
RSA rsa=new RSA();
DataInputStream in=new DataInputStream(System.in);
String testString;
System.out.println("Enter the plain text:");
testString=in.readLine();
System.out.println("Encrypting string:"+testString);
System.out.println("string in bytes:"+bytesToString(testString.getBytes()));
byte[] encrypted=rsa.encrypt(testString.getBytes());
byte[] decrypted=rsa.decrypt(encrypted);
System.out.println("Dcrypting Bytes:"+bytesToString(decrypted));
System.out.println("Dcrypted string:"+new String(decrypted));
}
private static String bytesToString(byte[] encrypted)
{
String test=" ";
for(byte b:encrypted)
{
test+=Byte.toString(b);
}
return test;
}
public byte[]encrypt(byte[]message)
{
return(new BigInteger(message)).modPow(e,N).toByteArray();
}
public byte[]decrypt(byte[]message)
{
return(new BigInteger(message)).modPow(d,N).toByteArray();
}
}
```

Output:

The screenshot displays a Windows desktop environment. A web browser window is open, showing a page titled "Program Output" from the URL `vtuloop.com/computer-network-lab-11/`. The page content shows the output of a Java program for RSA encryption. Below the browser window, a terminal window is open, showing the execution of the same Java program. The terminal output includes the generation of a public and private key, the encryption of a string, and the decryption of the resulting bytes. A red button with the text "Join For Job Alerts" and a WhatsApp icon is visible in the bottom right corner of the terminal window. The Windows taskbar at the bottom shows the Start button, a search bar, and several application icons. The system tray in the bottom right corner displays the date and time as 12:51 on 04-06-2024.

```
cse@cse:~$ gedit RSA.java
cse@cse:~$ javac RSA.java
Note: RSA.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
cse@cse:~$ java RSA
Prime number p is 1748931247591441976959575350759604195107021757572106694246514077177411950040460436614761678661412305782256321749981653657117
7526051919524133980921520311360621025940744801371543631243932937463237018914529433518257185751091632766692663281705794461840392310152013152321
054265934193649238556390112450664169032989
prime number q is 14720622357680992822813427150763115096267980256741457926440058417031520727146522800693980550532138495569779869495865517858950
5367749596146937804063973353014694599509478405104218694835854774629794951662431475697036034517995411560571267933885561096538830657418545954924
680615209946710684671741932414104704831971
Public key is 123670416810058759966055929883682546953822448944903568891263023009613028287226216395189756641502477788662940687201939899034118601
13875023959144721178489917
Private key is 20693813930906520750580099661259650745149754710924723402061154134757802381159173892908807605058225037406761360348293064919394445
67723994774094183851716782873690819130508394812285698528928056961545813906613330688673691530256226849087213040613247899386549722522543498918
7839836994111236804699775026007432912913842934242780608949470146287225886653770146609219432082157644828968166985670351545087397080623275521810
0817442744432755228125723010809550091194790672942241775860236178643028995990364897953565307279338790055798056914915298361667783421265821067533
873227459361202679909180128839032267639446204499180897501148053
Enter the plain text:
My bank password is Secret123
Encrypting string:My bank password is Secret123
string in bytes: 77121329897110107321129711511511911111410032105115328310199114101116495051
Decrypting Bytes: 77121329897110107321129711511511911111410032105115328310199114101116495051
Decrypted string:My bank password is Secret123
cse@cse:~$
```

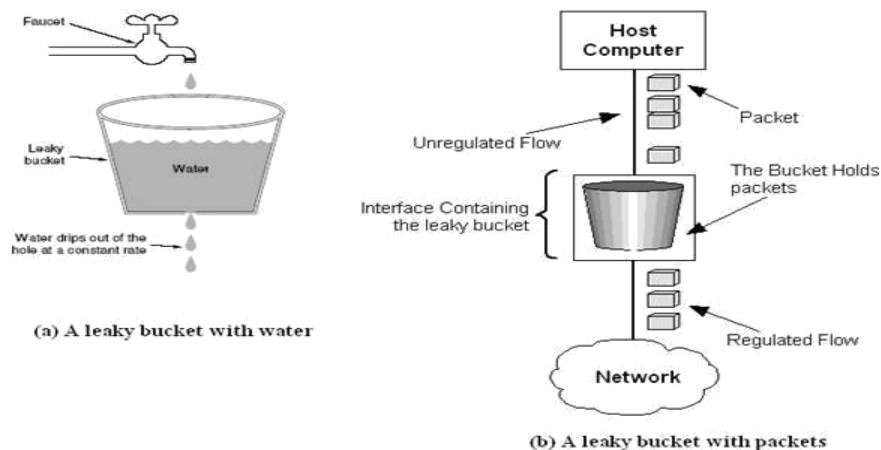
Experiment No: 8

Congestion Control Using Leaky Bucket Algorithm

Objective: Write a program for congestion control using leaky bucket algorithm.

Theory

The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero. From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).



While leaky bucket eliminates completely bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it doesn't take into account the idle process of the sender which means that if the host doesn't transmit data for some time the bucket becomes empty without permitting the transmission of any packet.

Algorithm

1. Start
2. Set the bucket size or the buffer size.
3. Set the output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)
6. Stop

Program

```
import java.util.Scanner;
import java.lang.*;
public class lab7
{
    public static void main(String[] args)
    {
        int i;
        int a[]=new int[20];
        int buck_rem=0,buck_cap=4,rate=3,sent,recv;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of packets");
        int n = in.nextInt();
        System.out.println("Enter the packets");
        for(i=1;i<=n;i++)
            a[i]= in.nextInt();
        System.out.println("Clock \t packet size \t accept \t sent \t remaining");
        for(i=1;i<=n;i++)
        {
            if(a[i]!=0)
            {
                if(buck_rem+a[i]>buck_cap)
                    recv=-1;
                else
                {
                    recv=a[i];
                    buck_rem+=a[i];
                }
            }
            else
                recv=0;
            if(buck_rem!=0)
            {
                if(buck_rem<rate)
                {
                    sent=buck_rem;
```

```
buck_rem=0;
}
else
{
sent=rate;
buck_rem=buck_rem-rate;
}
}
else
sent=0;
if(recv== -1)
System.out.println(+i+ "\t\t" +a[i]+ "\t dropped \t" + sent +"\t" +buck_rem);
else
System.out.println(+i+ "\t\t" +a[i] +"\t\t" +recv +"\t" +sent + "\t" +buck_rem);
}
}
}
```

Output:

Program Output

```
Enter the number of packets
5
Enter the packets
2
4
1
5
3
Clock   packet size   accept   sent   remaining
1       2           2        2       0
2       4           4        3       1
3       1           1        2       0
4       5       dropped    0       0
5       3       dropped    3       0
```