

# **LAB ASSIGNMENT-3**

## **CSN-361 Computer Networks Laboratory**

**Submitted by - Prateek Mali  
Enrollment no. - 17114059 (CSE)**

### **Problem Statements**

1. Write a socket program in C to determine class, Network and Host ID of an IPv4 address.
2. Write a C program to demonstrate File Transfer using UDP.
3. Write a TCL code for network simulator NS2 to demonstrate the star topology among a set of computer nodes. Given N nodes, one node will be assigned as the central node and the other nodes will be connected to it to form the star. You have to set up a TCP connection between k pairs of nodes and demonstrate the packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have

different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.

4. Write a TCL code for network simulator NS2 to demonstrate the ring topology among a set of computer nodes. Given N nodes, each node will be connected to two other nodes in the form of a ring. You have to set up a TCP connection between k pairs of nodes and demonstrate packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.
5. Write a TCL code for network simulator NS2 to demonstrate the bus topology among a set of computer nodes. Given N nodes, each node will be connected to a common link. You have to set up a TCP connection between k pairs of nodes and demonstrate packet transfer between them using Network Animator (NAM). Use File Transfer protocol (FTP) for the same. Each link should have different color of packets to differentiate the packets transferred between each pair of nodes. The program should take the number of nodes (N) as input followed by k pairs of nodes.

# 1

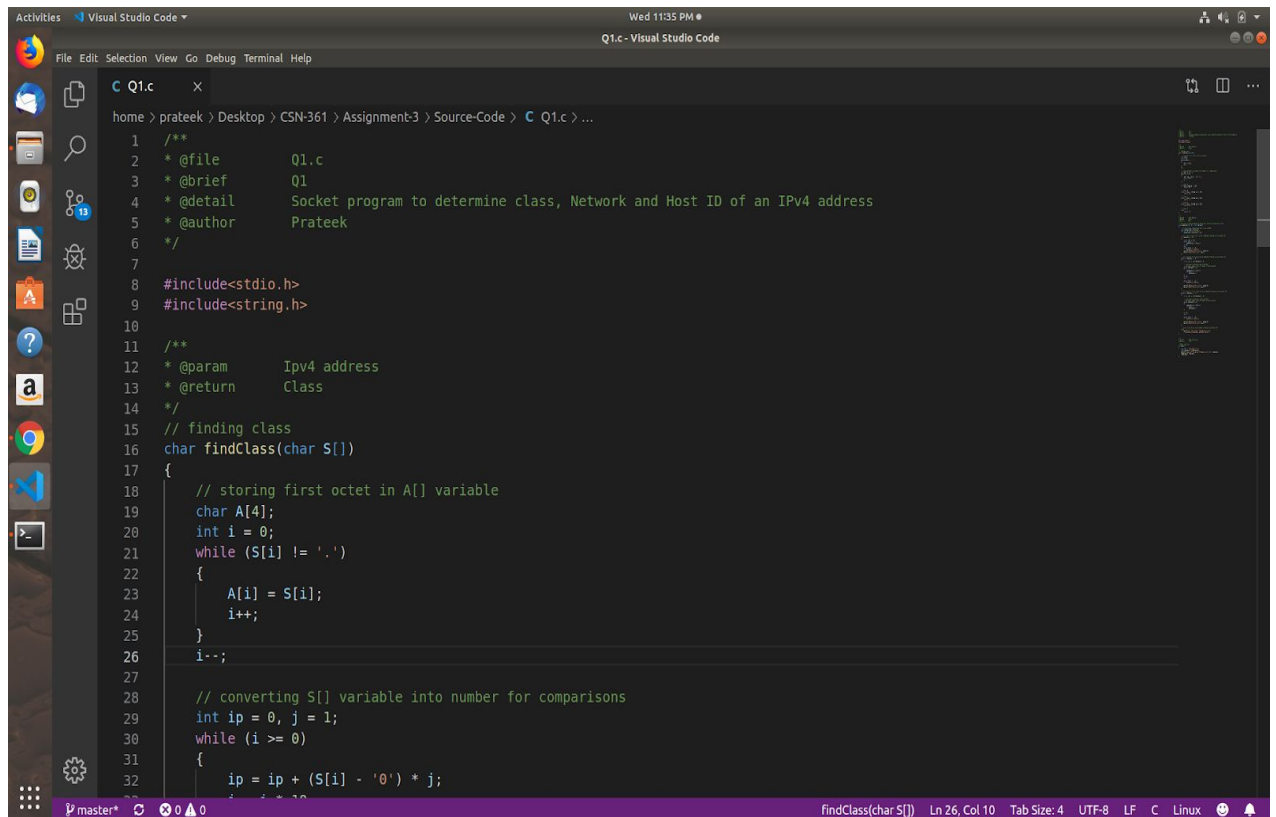
## **Implementations details**

1. For determining the class we checked first octet of IP address. As we know, for class A first octet will range from 1 – 126, for class B first octet will range from 128 – 191, for class C first octet will range from 192-223, for class D first octet will range from 224 – 239, for class E first octet will range from 240 – 255.
2. For determining the Network and Host ID: We know that Subnet Mask for Class A is 8, for Class B is 16 and for Class C is 24 whereas Class D and E is not divided into Network and Host ID.

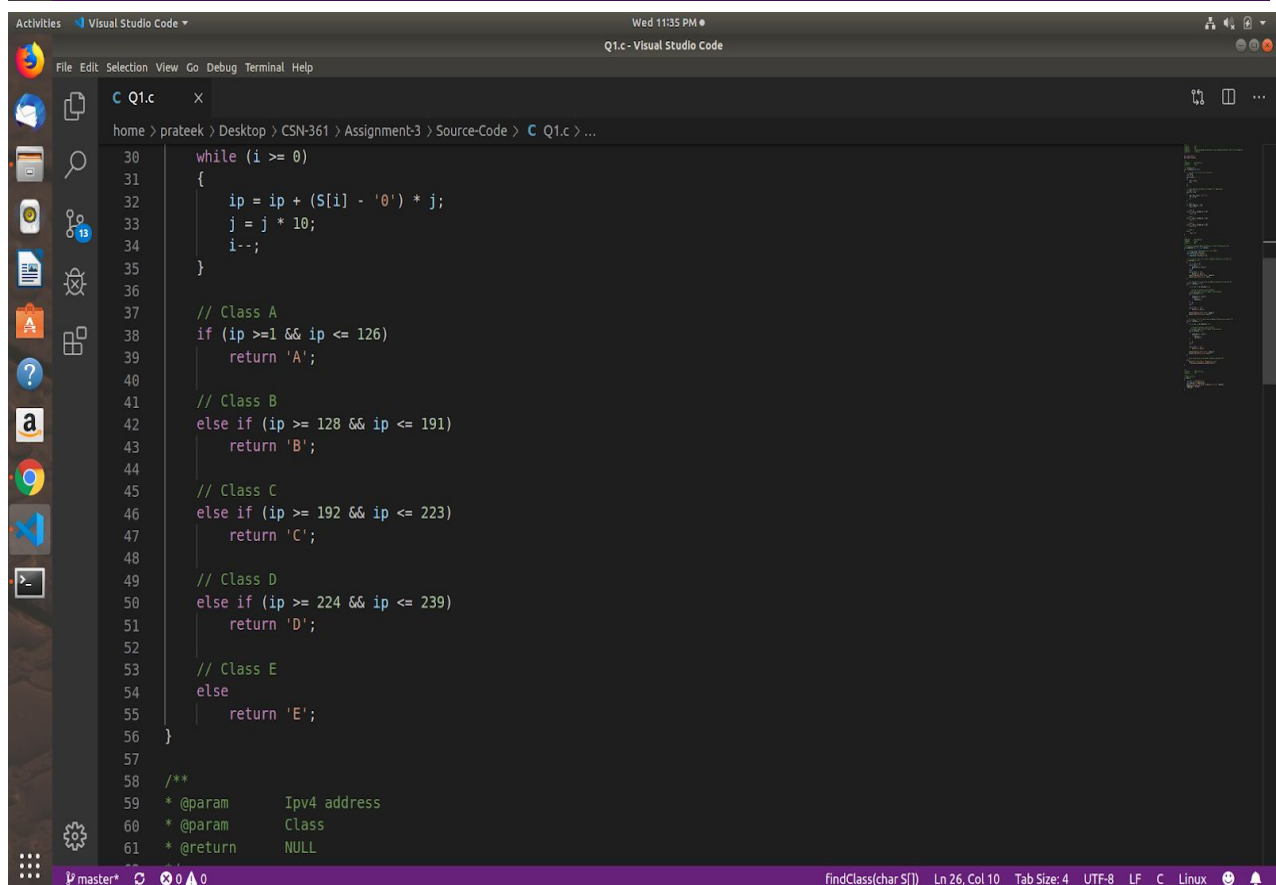
## **Data Structures used**

1. Array has been used as the major data structure.

# Code Snippets



```
1  /**
2  * @file      Q1.c
3  * @brief     Q1
4  * @detail    Socket program to determine class, Network and Host ID of an IPv4 address
5  * @author    Prateek
6  */
7
8  #include<stdio.h>
9  #include<string.h>
10
11  /**
12  * @param      Ipv4 address
13  * @return     Class
14  */
15  // finding class
16  char findClass(char S[])
17  {
18      // storing first octet in A[] variable
19      char A[4];
20      int i = 0;
21      while (S[i] != '.')
22      {
23          A[i] = S[i];
24          i++;
25      }
26      i--;
27
28      // converting S[] variable into number for comparisons
29      int ip = 0, j = 1;
30      while (i >= 0)
31      {
32          ip = ip + (S[i] - '0') * j;
```



```
30      while (i >= 0)
31      {
32          ip = ip + (S[i] - '0') * j;
33          j = j * 10;
34          i--;
35      }
36
37      // Class A
38      if (ip >= 1 && ip <= 126)
39          return 'A';
40
41      // Class B
42      else if (ip >= 128 && ip <= 191)
43          return 'B';
44
45      // Class C
46      else if (ip >= 192 && ip <= 223)
47          return 'C';
48
49      // Class D
50      else if (ip >= 224 && ip <= 239)
51          return 'D';
52
53      // Class E
54      else
55          return 'E';
56  }
57
58  /**
59  * @param      Ipv4 address
60  * @param      Class
61  * @return     NULL
```

```
Visual Studio Code
Wed 11:36 PM
Q1.c - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

C Q1.c
home > prateek > Desktop > CSN-361 > Assignment-3 > Source-Code > C Q1.c > ...

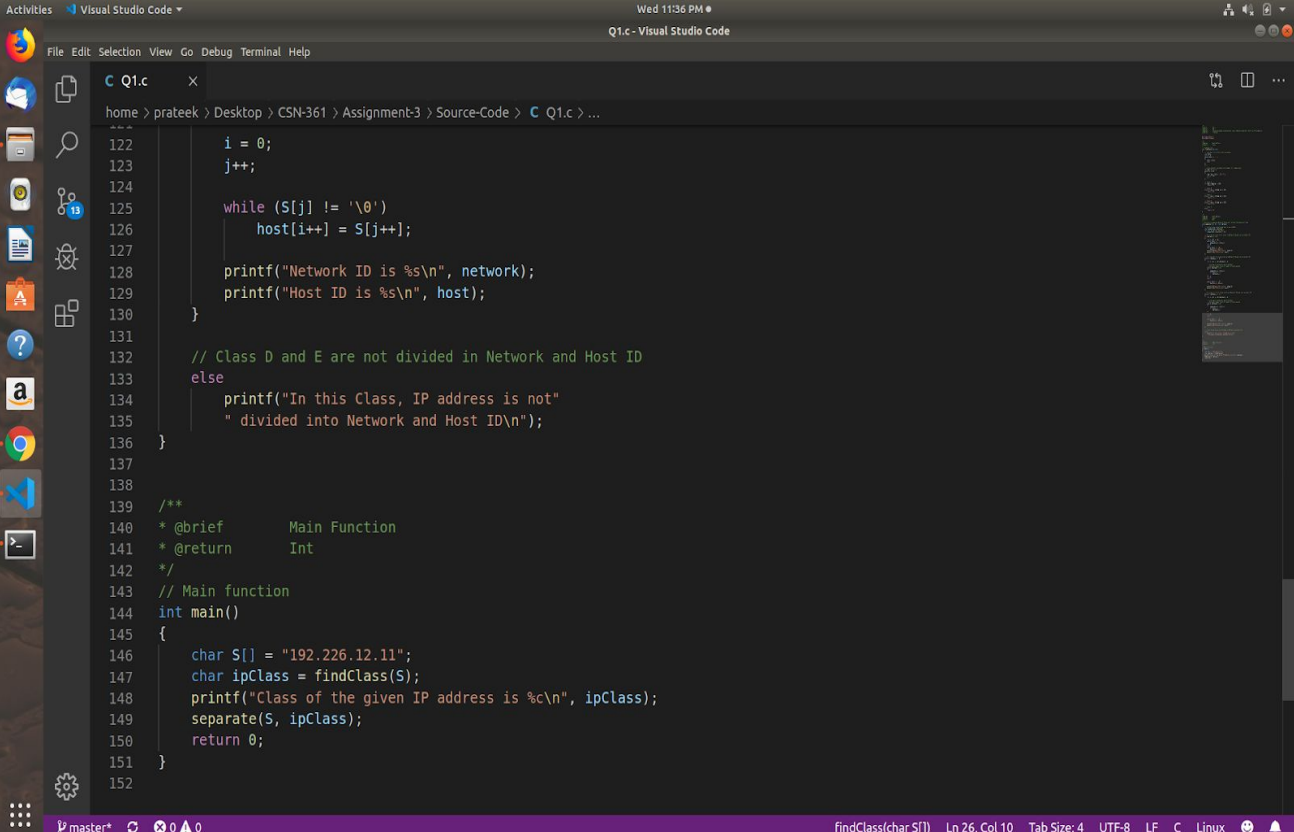
59 * @param Ipv4 address
60 * @param Class
61 * @return NULL
62 */
63 // Function to separate Network ID as well as Host ID and print them
64 void separate(char S[], char ipClass)
65 {
66     // Initializing network and host array to NULL
67     char network[12], host[12];
68     for (int k = 0; k < 12; k++)
69         network[k] = host[k] = '\0';
70
71     // for class A, only first octet is Network ID and rest are Host ID
72     if (ipClass == 'A')
73     {
74         int i = 0, j = 0;
75         while (S[j] != '.')
76             network[i++] = S[j++];
77         i = 0;
78         j++;
79         while (S[j] != '\0')
80             host[i++] = S[j++];
81         printf("Network ID is %s\n", network);
82         printf("Host ID is %s\n", host);
83     }
84
85     // for class B, first two octet are Network ID and rest are Host ID
86     else if (ipClass == 'B')
87     {
88         int i = 0, j = 0, dotCount = 0;
89
90         // storing in network[] up to 2nd dot
```

```
Visual Studio Code
Wed 11:36 PM
Q1.c - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

C Q1.c
home > prateek > Desktop > CSN-361 > Assignment-3 > Source-Code > C Q1.c > ...

90         // storing in network[] up to 2nd dot
91         // dotCount keeps track of number of dots passed
92         while (dotCount < 2)
93         {
94             network[i++] = S[j++];
95             if (S[j] == '.')
96                 dotCount++;
97         }
98         i = 0;
99         j++;
100
101         while (S[j] != '\0')
102             host[i++] = S[j++];
103
104         printf("Network ID is %s\n", network);
105         printf("Host ID is %s\n", host);
106     }
107
108     // for class C, first three octet are Network ID and rest are Host ID
109     else if (ipClass == 'C')
110     {
111         int i = 0, j = 0, dotCount = 0;
112
113         // storing in network[] up to 3rd dot
114         // dotCount keeps track of number of dots passed
115         while (dotCount < 3)
116         {
117             network[i++] = S[j++];
118             if (S[j] == '.')
119                 dotCount++;
120         }
121
122         i = 0;
```

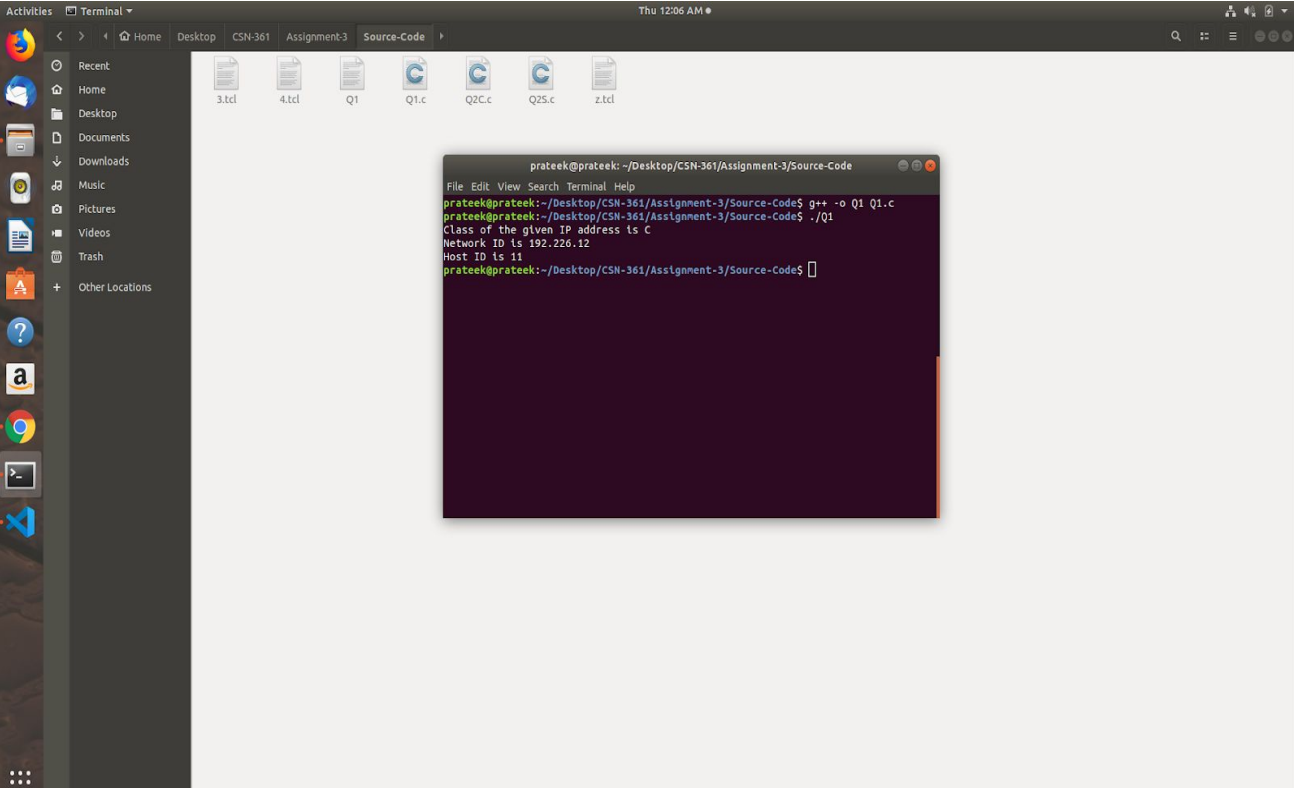


The screenshot shows the Visual Studio Code editor with a C file named `Q1.c` open. The code is a C program that determines the class of an IP address and prints the network and host IDs. The code is as follows:

```
122     i = 0;
123     j++;
124
125     while (S[j] != '\0')
126     {
127         host[i++] = S[j++];
128
129         printf("Network ID is %s\n", network);
130         printf("Host ID is %s\n", host);
131     }
132
133     // Class D and E are not divided in Network and Host ID
134     else
135     {
136         printf("In this Class, IP address is not"
137             " divided into Network and Host ID\n");
138     }
139
140     /**
141     * @brief      Main Function
142     * @return     Int
143     */
144     // Main function
145     int main()
146     {
147         char S[] = "192.226.12.11";
148         char ipClass = findClass(S);
149         printf("Class of the given IP address is %c\n", ipClass);
150         separate(S, ipClass);
151         return 0;
152     }
```

The status bar at the bottom indicates the current position is at line 26, column 10, with a tab size of 4, UTF-8 encoding, LF line endings, and the C language on a Linux system.

## Snapshot of running code



The screenshot shows a terminal window with the following commands and output:

```
prateek@prateek: ~/Desktop/CSN-361/Assignment-3/Source-Code
prateek@prateek:~/Desktop/CSN-361/Assignment-3/Source-Code$ g++ -o Q1 Q1.c
prateek@prateek:~/Desktop/CSN-361/Assignment-3/Source-Code$ ./Q1
Class of the given IP address is C
Network ID is 192.226.12
Host ID is 11
prateek@prateek:~/Desktop/CSN-361/Assignment-3/Source-Code$
```

The terminal output matches the expected results from the C code: the IP class is 'C', the network ID is '192.226.12', and the host ID is '11'.

## **2**

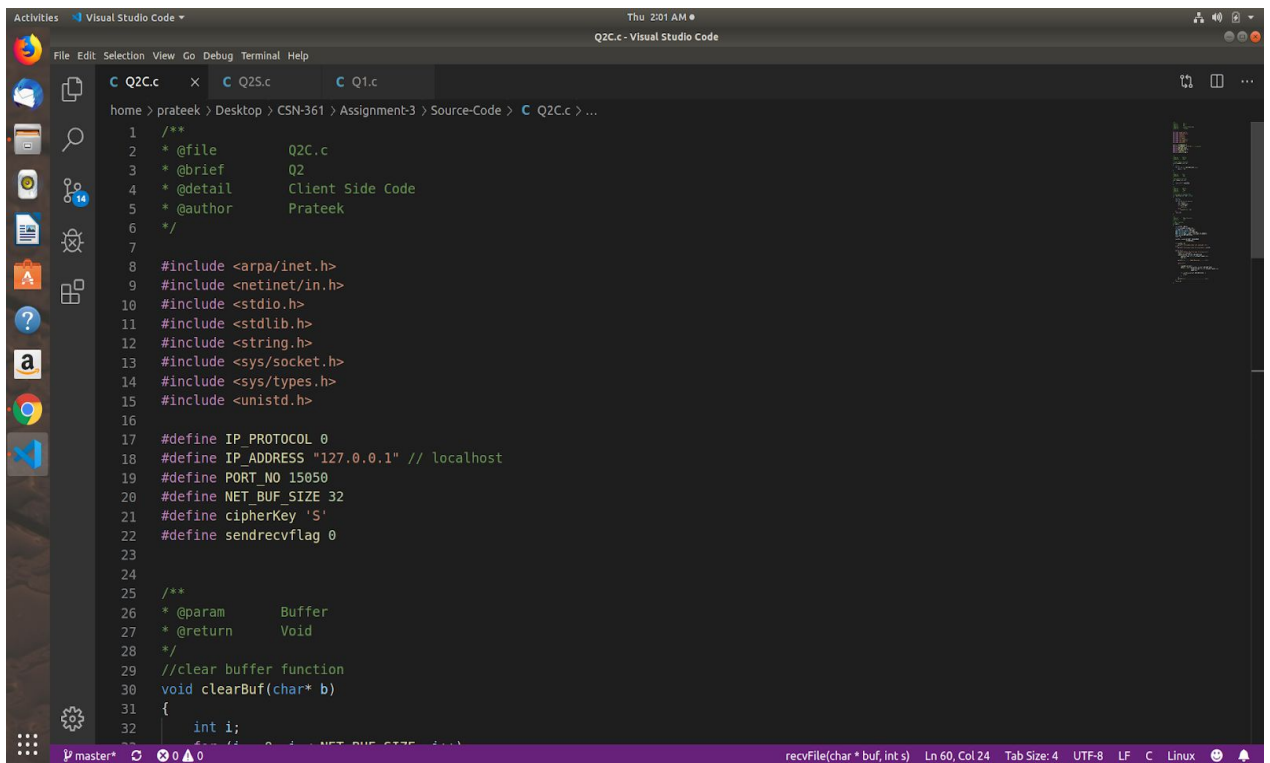
### **Implementations details (Algorithm)**

1. The server starts and waits for filename.
2. The client sends a filename.
3. The server receives filename. If file is present, server starts reading file and continues to send a buffer filled with file contents encrypted until file-end is reached.
4. End is marked by EOF.
5. File is received as buffers until EOF is received. Then it is decrypted.
6. If Not present, a file not found is sent.

### **Data Structures used**

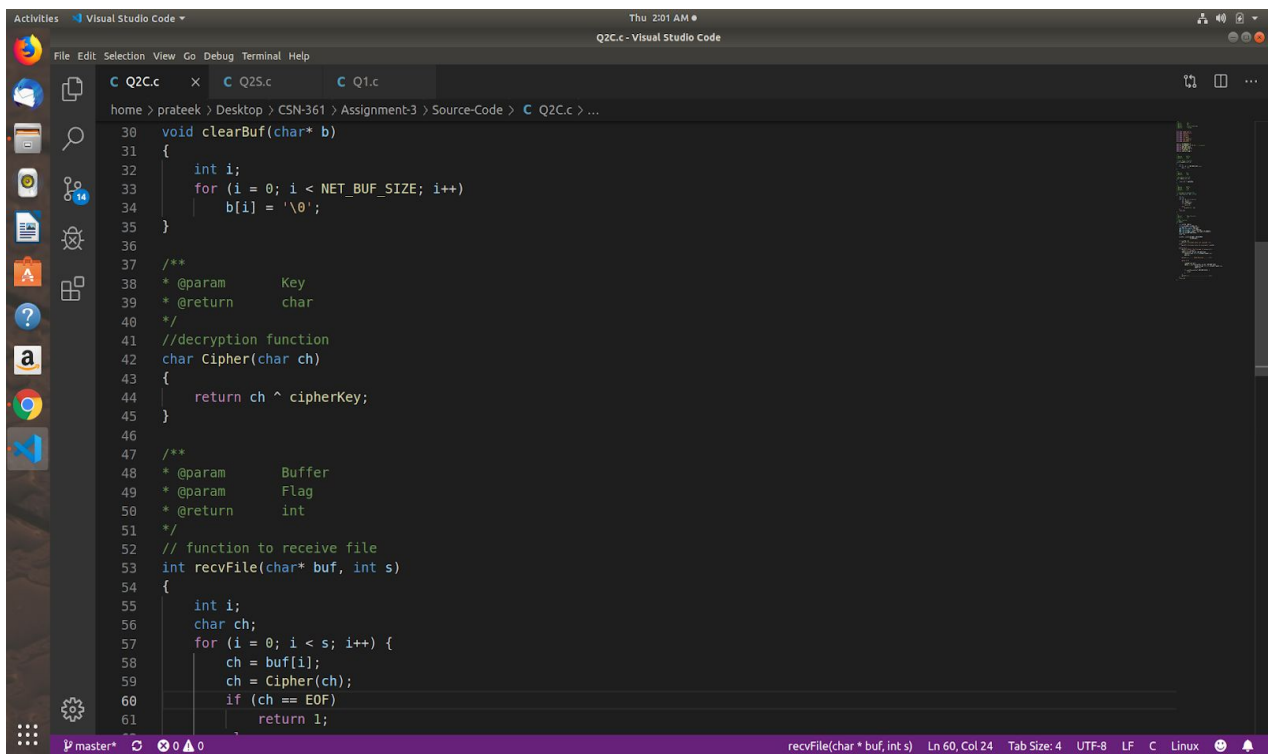
1. Buffer has been used as the major data structure.

### **Client-Side Code Snippets**



Visual Studio Code interface showing the file `Q2C.c`. The code is a C program for a client-side network application. It includes standard headers like `arpa/inet.h`, `netinet/in.h`, `stdio.h`, `stdlib.h`, `string.h`, `sys/socket.h`, `sys/types.h`, and `unistd.h`. It defines constants for IP protocol, address, port, buffer size, cipher key, and a flag. The `clearBuf` function is defined to clear a buffer.

```
1  /**
2  * @file      Q2C.c
3  * @brief     Q2
4  * @detail    Client Side Code
5  * @author    Prateek
6  */
7
8  #include <arpa/inet.h>
9  #include <netinet/in.h>
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13 #include <sys/socket.h>
14 #include <sys/types.h>
15 #include <unistd.h>
16
17 #define IP_PROTOCOL 0
18 #define IP_ADDRESS "127.0.0.1" // localhost
19 #define PORT_NO 15050
20 #define NET_BUF_SIZE 32
21 #define cipherKey 'S'
22 #define sendrecvflag 0
23
24
25 /**
26 * @param      Buffer
27 * @return     Void
28 */
29 //clear buffer function
30 void clearBuf(char* b)
31 {
32     int i;
```



Visual Studio Code interface showing the continuation of the file `Q2C.c`. The code continues with the `clearBuf` function implementation, a `Cipher` function for encryption, a `decryption` function, and the start of the `recvFile` function which receives data from a socket.

```
30 void clearBuf(char* b)
31 {
32     int i;
33     for (i = 0; i < NET_BUF_SIZE; i++)
34         b[i] = '\0';
35 }
36
37 /**
38 * @param      Key
39 * @return     char
40 */
41 //decryption function
42 char Cipher(char ch)
43 {
44     return ch ^ cipherKey;
45 }
46
47 /**
48 * @param      Buffer
49 * @param      Flag
50 * @return     int
51 */
52 // function to receive file
53 int recvFile(char* buf, int s)
54 {
55     int i;
56     char ch;
57     for (i = 0; i < s; i++) {
58         ch = buf[i];
59         ch = Cipher(ch);
60         if (ch == EOF)
61             return 1;
```



Visual Studio Code - Thu 2:01 AM \*

File Edit Selection View Go Debug Terminal Help

C Q2C.c x C Q2S.c C Q1.c

home > prateek > Desktop > CSN-361 > Assignment-3 > Source-Code > C Q2C.c > ...

```
62         else
63             printf("%c", ch);
64     }
65     return 0;
66 }
67
68 /**
69  * @brief      Main Function
70  * @return     Int
71  */
72 // Main function
73 int main()
74 {
75     int sockfd, nBytes;
76     struct sockaddr_in add;
77     int addrlen = sizeof(a PF_INET
78     addr_con.sin_family = AF_INET;
79     addr_con.sin_port = htons(PORT_NO);
80     addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
81     char net_buf[NET_BUF_SIZE];
82     FILE* fp;
83
84     sockfd = socket(AF_INET, SOCK_DGRAM,
85                     IP_PROTOCOL);
86
87     if (sockfd < 0)
88         printf("\nfile descriptor not received!!\n");
89     else
90         printf("\nfile descriptor %d received\n", sockfd);
91
92     while (1) {
93         printf("\nEnter the file name to receive:\n");
```

recvfFile(char \* buf, int s) Ln 60, Col 24 Tab Size: 4 UTF-8 LF C Linux

Visual Studio Code - Thu 2:01 AM \*

File Edit Selection View Go Debug Terminal Help

C Q2C.c x C Q2S.c C Q1.c

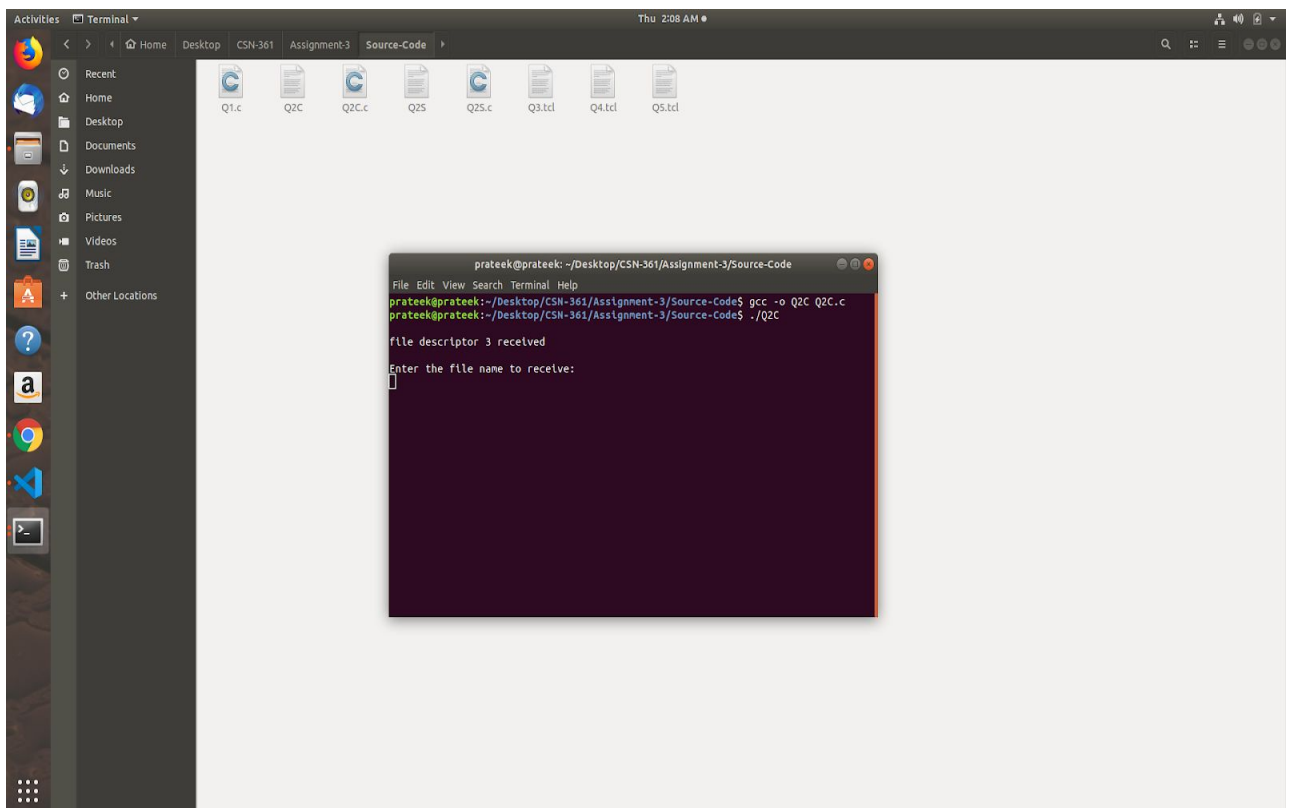
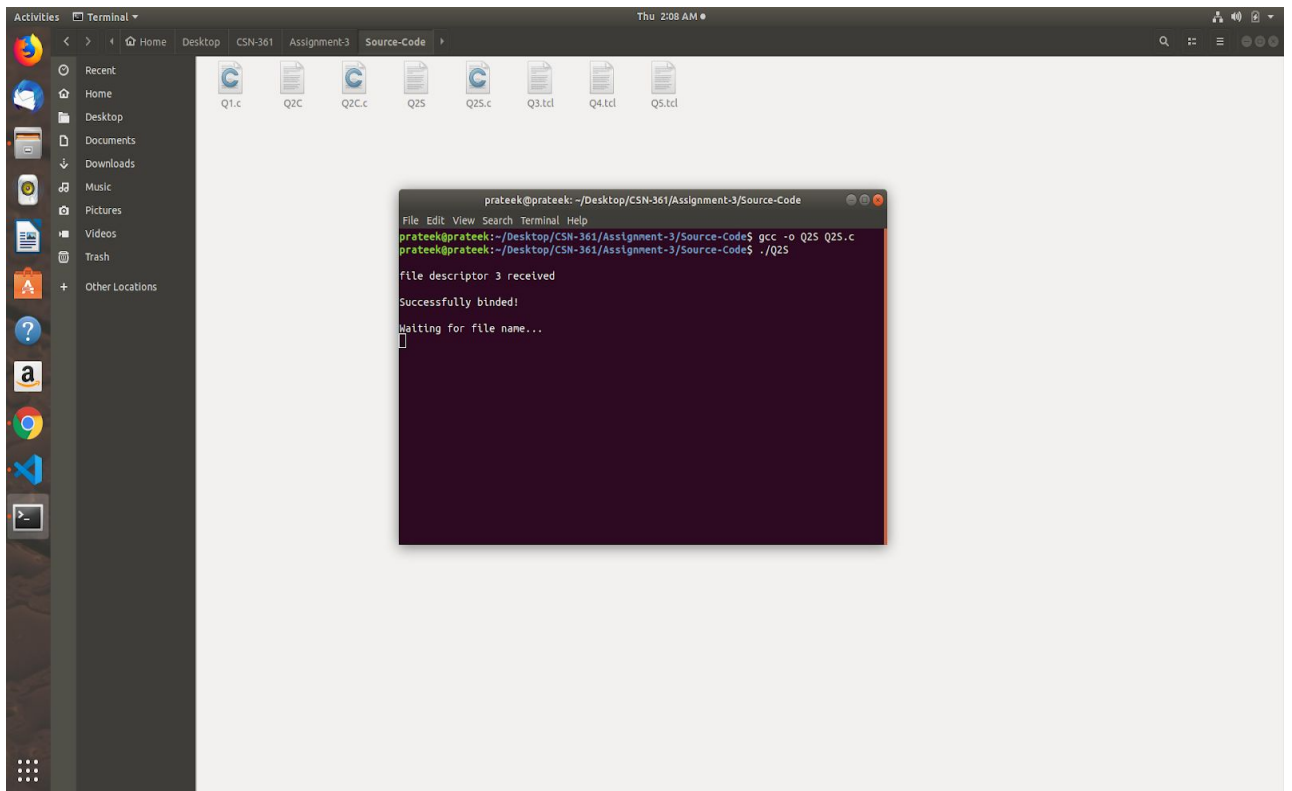
home > prateek > Desktop > CSN-361 > Assignment-3 > Source-Code > C Q2C.c > ...

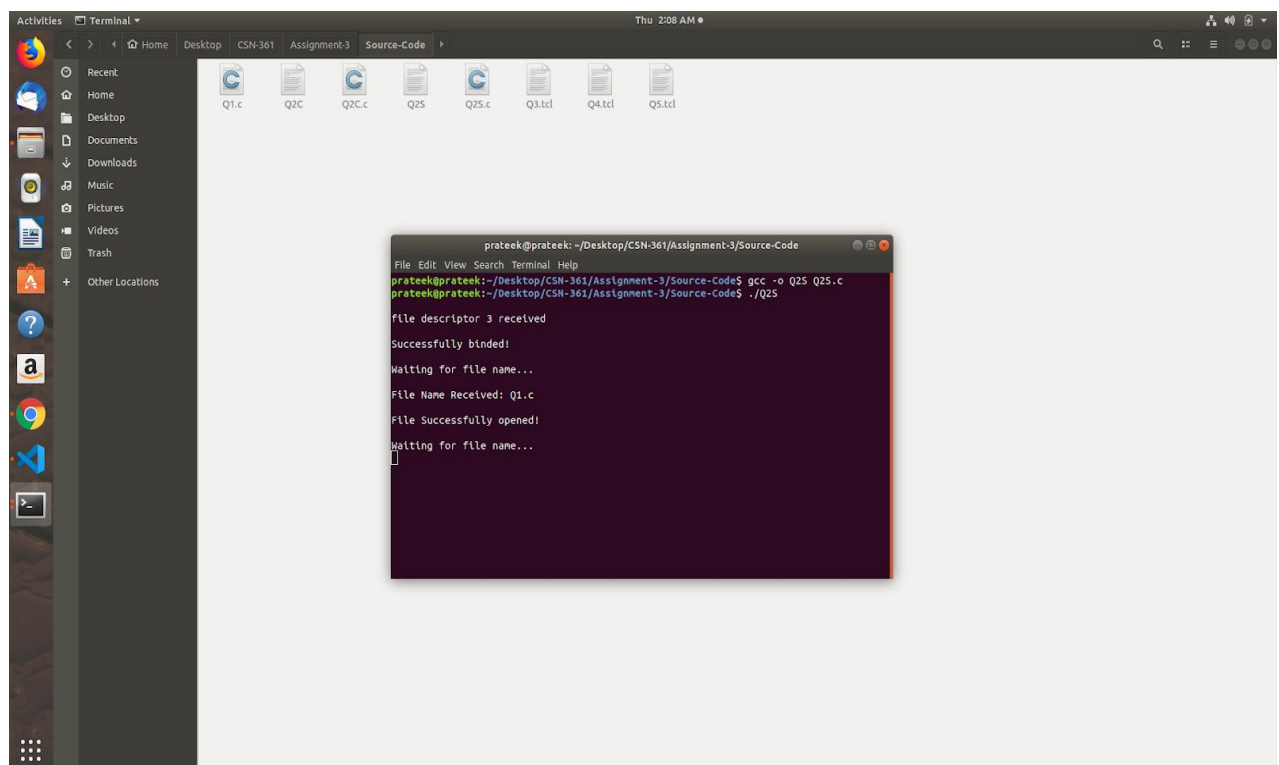
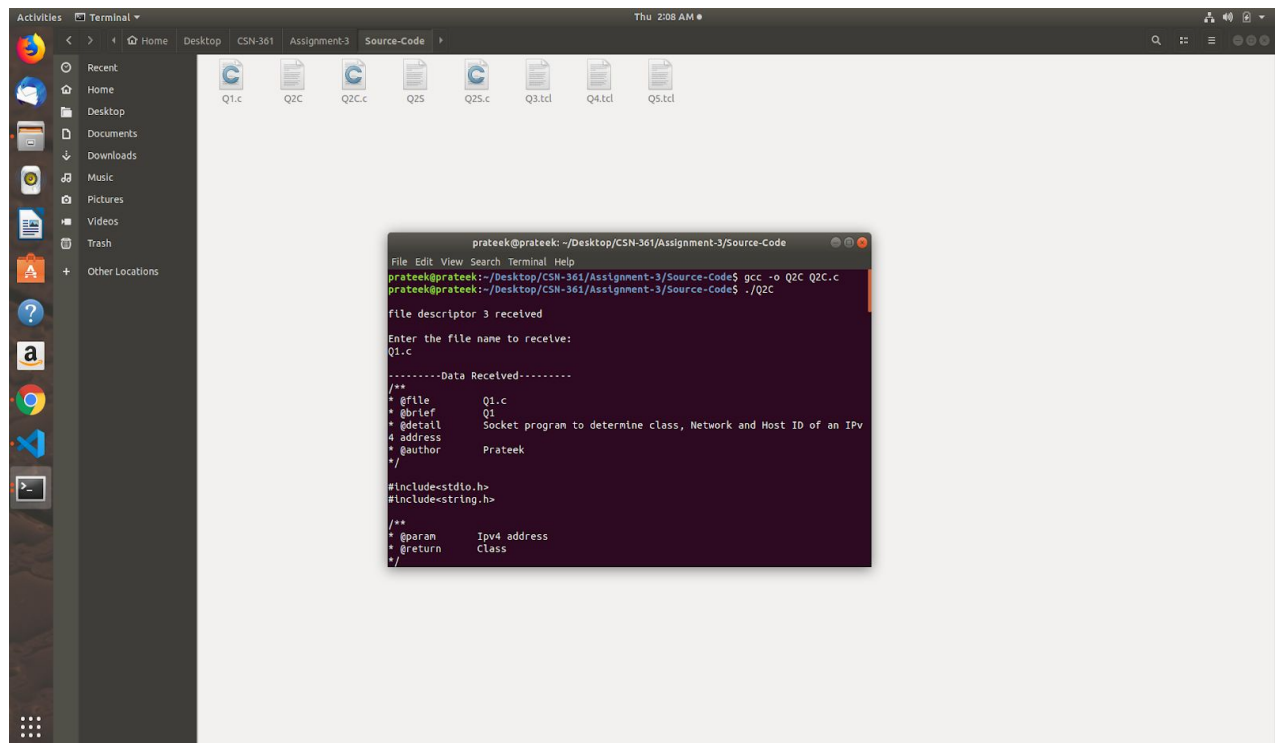
```
88     printf("\nfile descriptor not received!!\n");
89     else
90         printf("\nfile descriptor %d received\n", sockfd);
91
92     while (1) {
93         printf("\nEnter the file name to receive:\n");
94         scanf("%s", net_buf);
95         sendto(sockfd, net_buf, NET_BUF_SIZE,
96               sendrecvflag, (struct sockaddr*)&addr_con,
97                           addrlen);
98
99         printf("\n-----Data Received-----\n");
100
101         while (1) {
102             #define sendrecvflag 0
103             clearBuf(net_buf);
104             nBytes = recvfro 0
105                     sendrecvflag, (struct sockaddr*)&addr_con,
106                               &addrlen);
107
108             if (recvFile(net_buf, NET_BUF_SIZE)) {
109                 break;
110             }
111         }
112         printf("\n-----\n");
113     }
114     return 0;
115 }
116
```

recvfFile(char \* buf, int s) Ln 60, Col 24 Tab Size: 4 UTF-8 LF C Linux



## Snapshots of running code



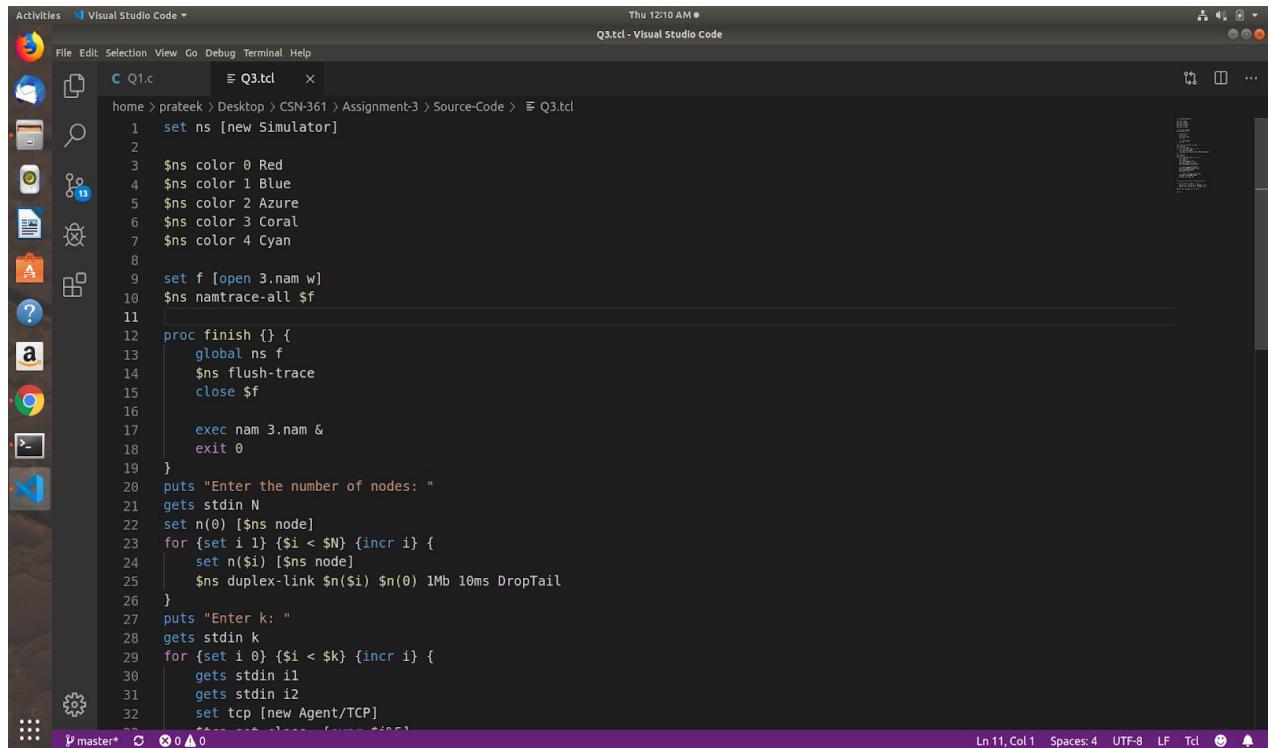


**3**

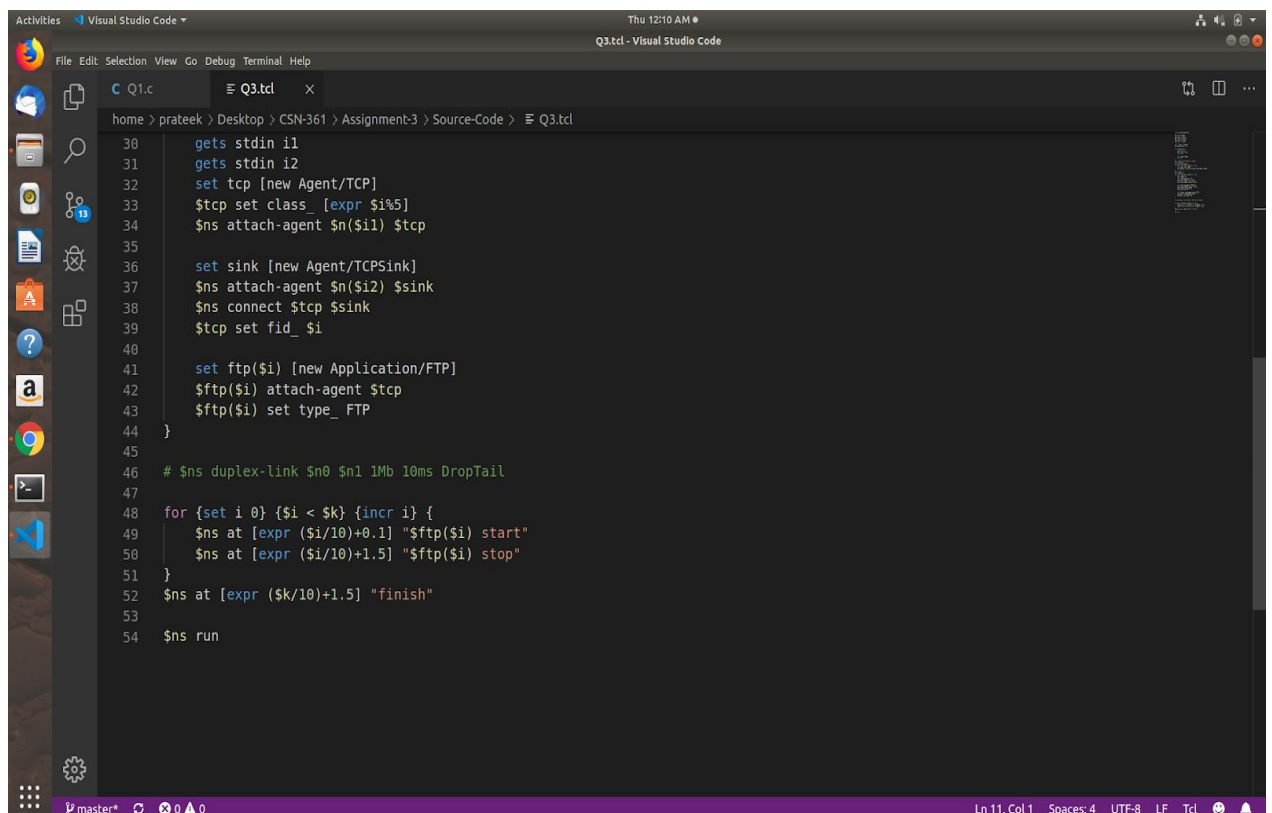
## **Implementations details**

We have implemented star topology. A star topology is a topology for a Local Area Network (LAN) in which all nodes are individually connected to a central connection point, like a hub or a switch.

## Code Snippets

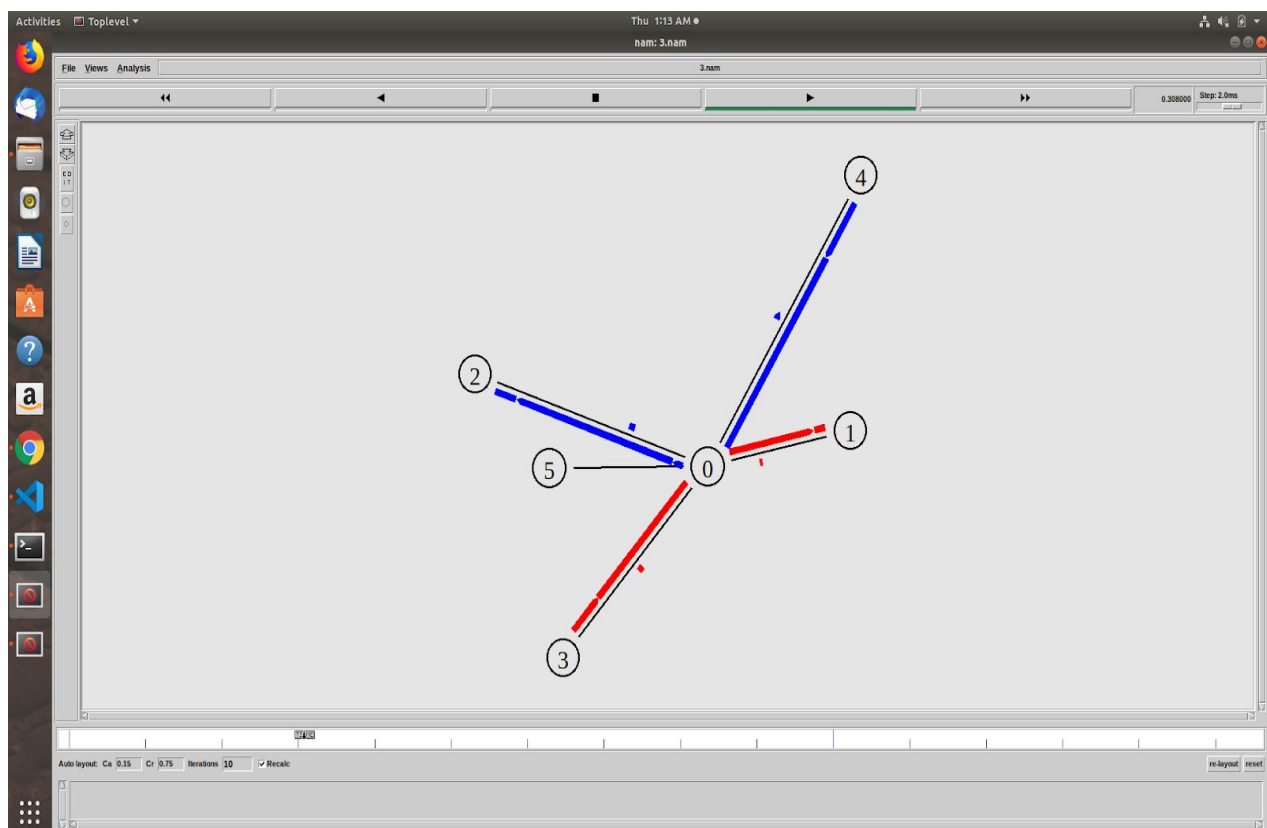
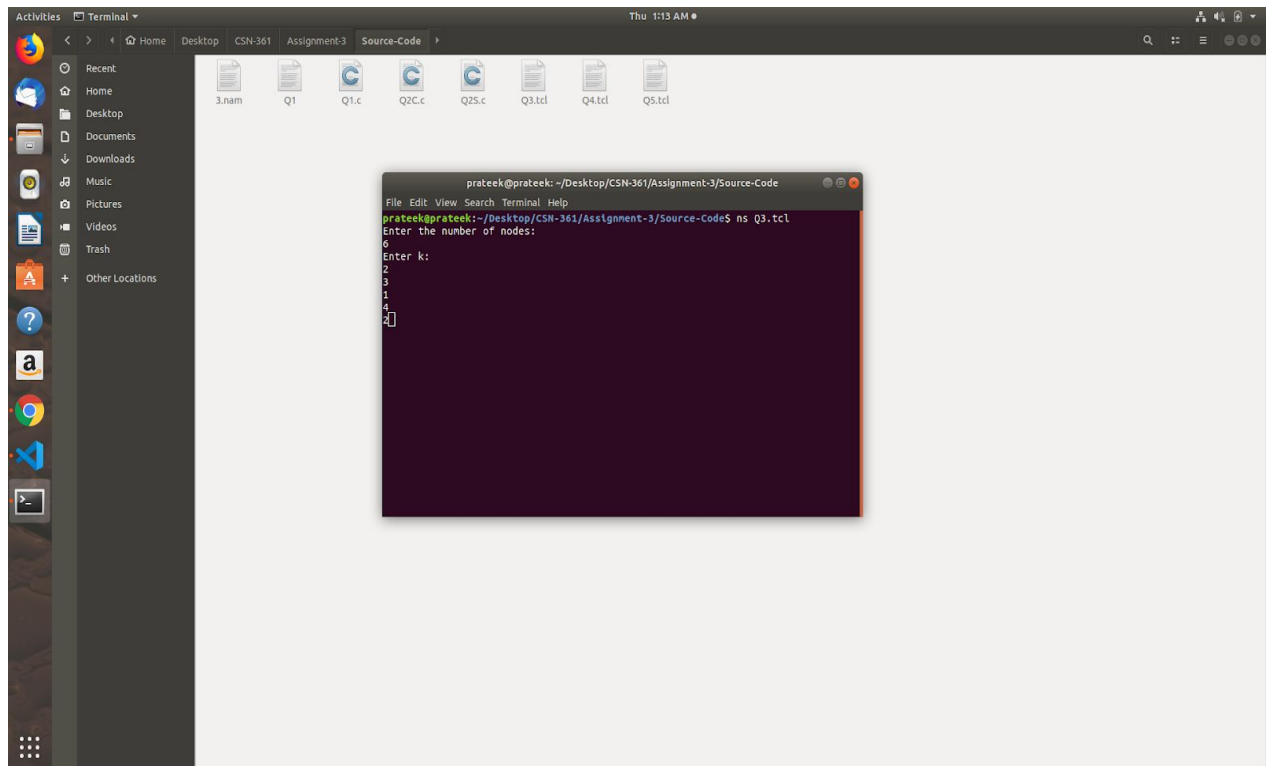


```
1 set ns [new Simulator]
2
3 $ns color 0 Red
4 $ns color 1 Blue
5 $ns color 2 Azure
6 $ns color 3 Coral
7 $ns color 4 Cyan
8
9 set f [open 3.nam w]
10 $ns namtrace-all $f
11
12 proc finish {} {
13     global ns f
14     $ns flush-trace
15     close $f
16
17     exec nam 3.nam &
18     exit 0
19 }
20 puts "Enter the number of nodes: "
21 gets stdin N
22 set n(0) [$ns node]
23 for {set i 1} {$i < $N} {incr i} {
24     set n($i) [$ns node]
25     $ns duplex-link $n($i) $n(0) 1Mb 10ms DropTail
26 }
27 puts "Enter k: "
28 gets stdin k
29 for {set i 0} {$i < $k} {incr i} {
30     gets stdin i1
31     gets stdin i2
32     set tcp [new Agent/TCP]
```



```
33     $tcp set class_ [expr $i%5]
34     $ns attach-agent $n($i1) $tcp
35
36     set sink [new Agent/TCPSink]
37     $ns attach-agent $n($i2) $sink
38     $ns connect $tcp $sink
39     $tcp set fid_ $i
40
41     set ftp($i) [new Application/FTP]
42     $ftp($i) attach-agent $tcp
43     $ftp($i) set type_FTP
44 }
45
46 # $ns duplex-link $n0 $n1 1Mb 10ms DropTail
47
48 for {set i 0} {$i < $k} {incr i} {
49     $ns at [expr ($i/10)+0.1] "$ftp($i) start"
50     $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
51 }
52 $ns at [expr ($k/10)+1.5] "finish"
53
54 $ns run
```

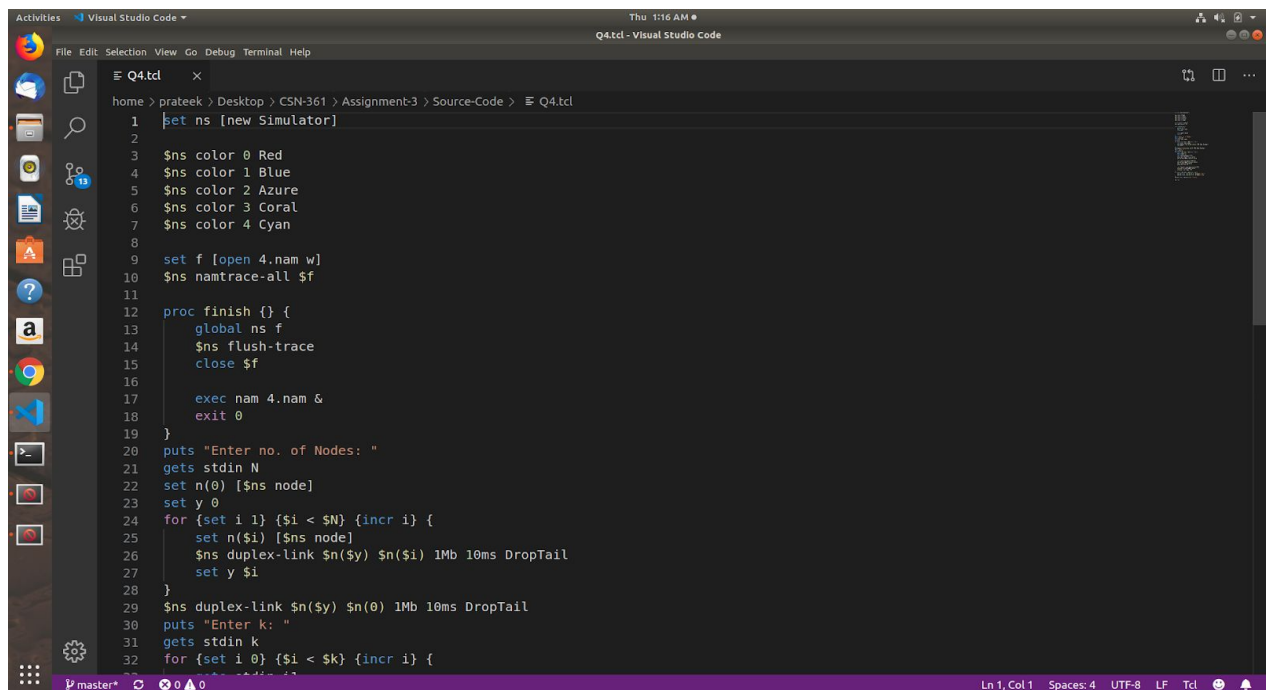
## Snapshots of running code



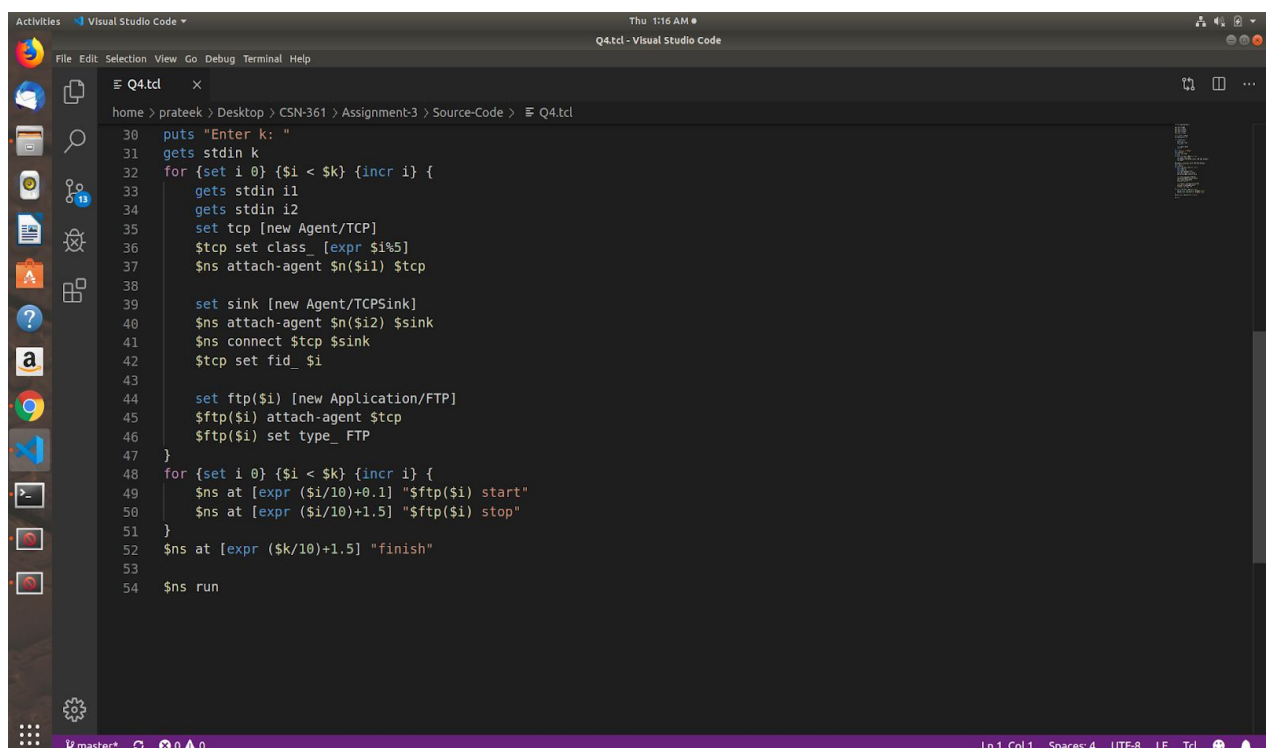
## Implementations details

We have implemented ring topology. A ring topology is a network configuration in which device connections create a circular data path. Each networked device is connected to two others, like points on a circle.

## Code Snippets

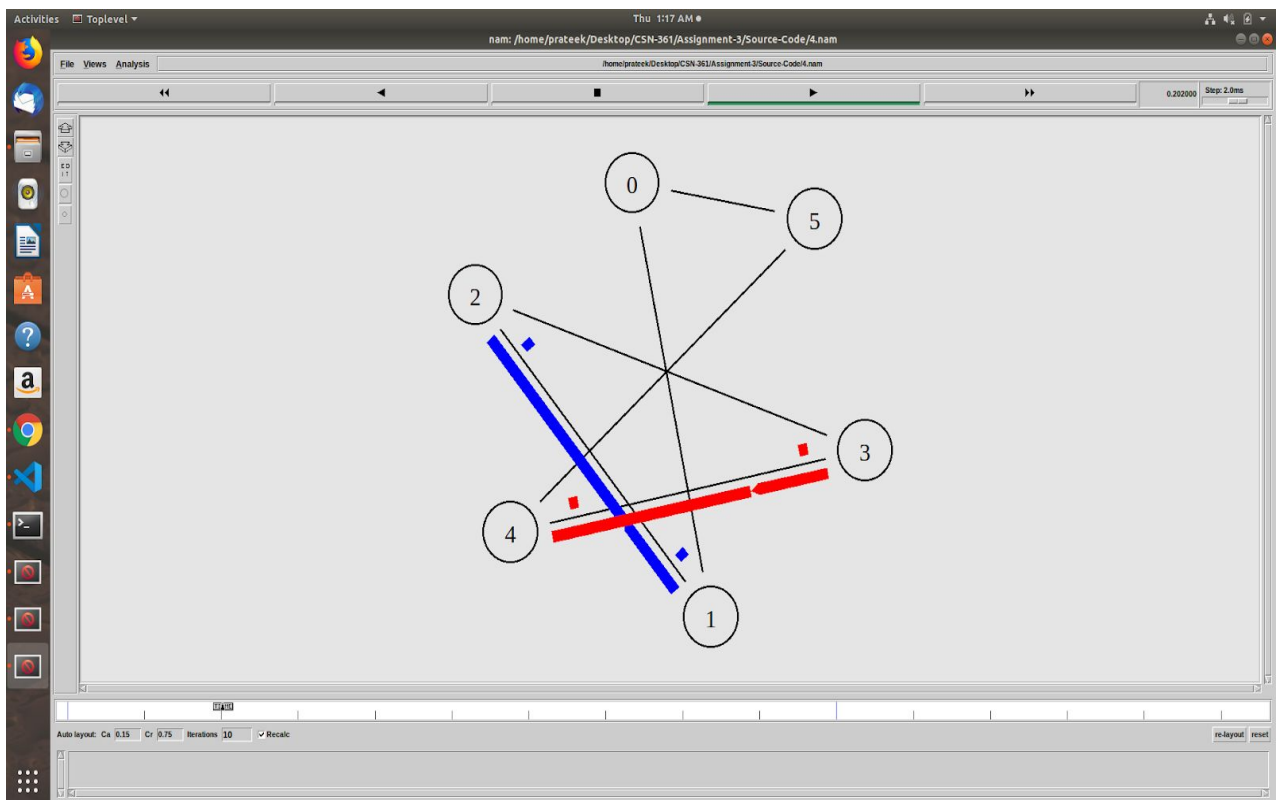
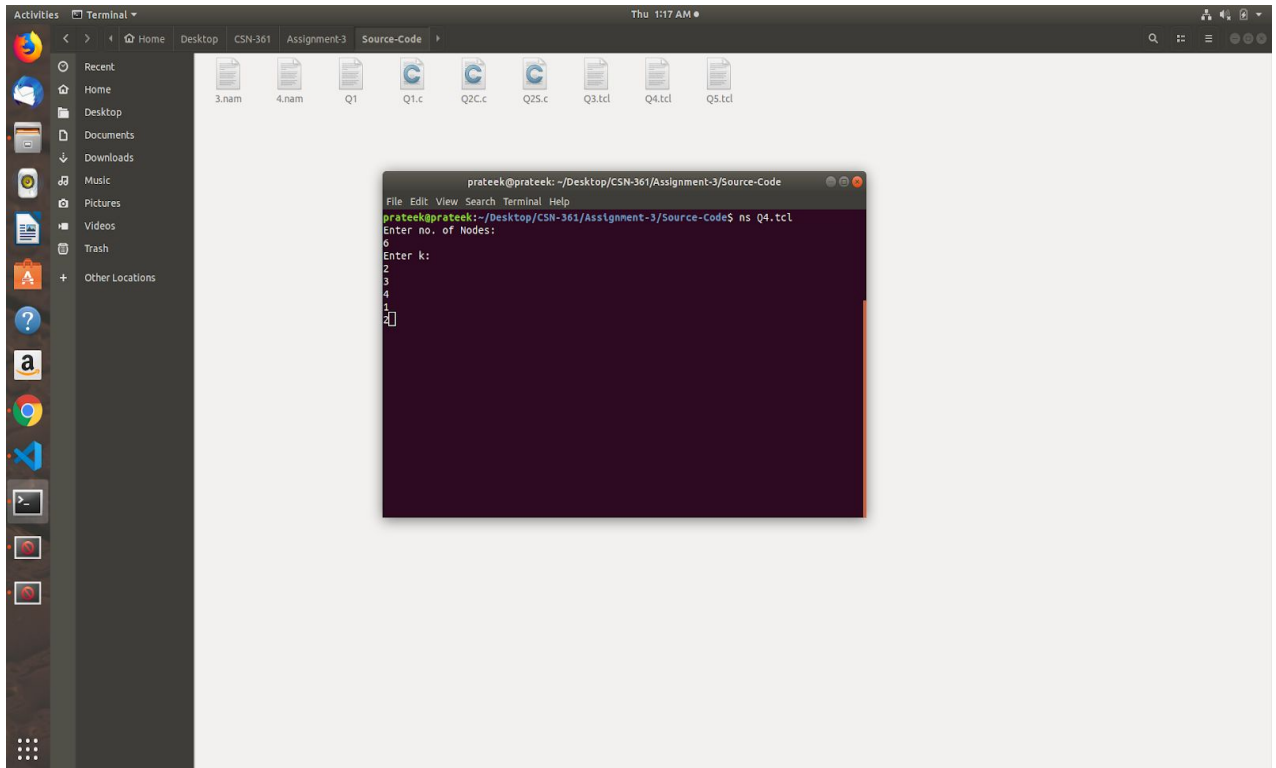


```
1 set ns [new Simulator]
2
3 $ns color 0 Red
4 $ns color 1 Blue
5 $ns color 2 Azure
6 $ns color 3 Coral
7 $ns color 4 Cyan
8
9 set f [open 4.nam w]
10 $ns namtrace-all $f
11
12 proc finish {} {
13     global ns f
14     $ns flush-trace
15     close $f
16
17     exec nam 4.nam &
18     exit 0
19 }
20 puts "Enter no. of Nodes: "
21 gets stdin N
22 set n(0) [$ns node]
23 set y 0
24 for {set i 1} {$i < $N} {incr i} {
25     set n($i) [$ns node]
26     $ns duplex-link $n($y) $n($i) 1Mb 10ms DropTail
27     set y $i
28 }
29 $ns duplex-link $n($y) $n(0) 1Mb 10ms DropTail
30 puts "Enter k: "
31 gets stdin k
32 for {set i 0} {$i < $k} {incr i} {
```



```
30 puts "Enter k: "
31 gets stdin k
32 for {set i 0} {$i < $k} {incr i} {
33     gets stdin i1
34     gets stdin i2
35     set tcp [new Agent/TCP]
36     $tcp set class_ [expr $i%5]
37     $ns attach-agent $n($i1) $tcp
38
39     set sink [new Agent/TCPSink]
40     $ns attach-agent $n($i2) $sink
41     $ns connect $tcp $sink
42     $tcp set fid_ $i
43
44     set ftp($i) [new Application/FTP]
45     $ftp($i) attach-agent $tcp
46     $ftp($i) set type_ FTP
47 }
48 for {set i 0} {$i < $k} {incr i} {
49     $ns at [expr ($i/10)+0.1] "$ftp($i) start"
50     $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
51 }
52 $ns at [expr ($k/10)+1.5] "finish"
53
54 $ns run
```

## Snapshots of running code

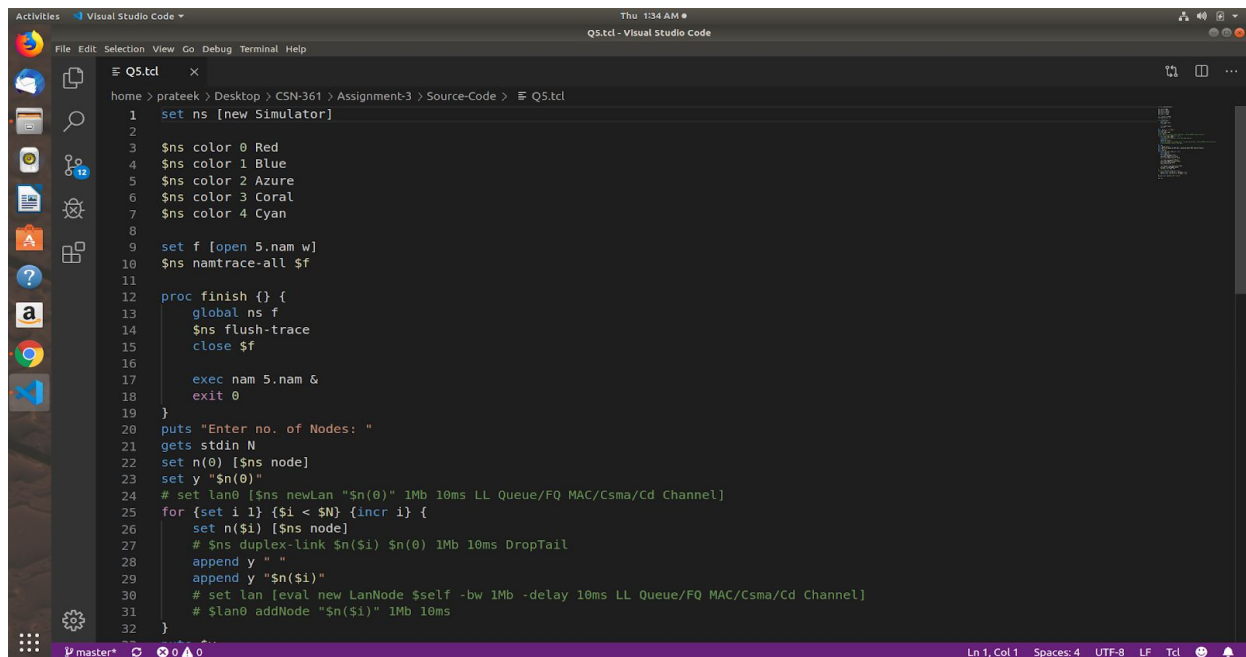


# 5

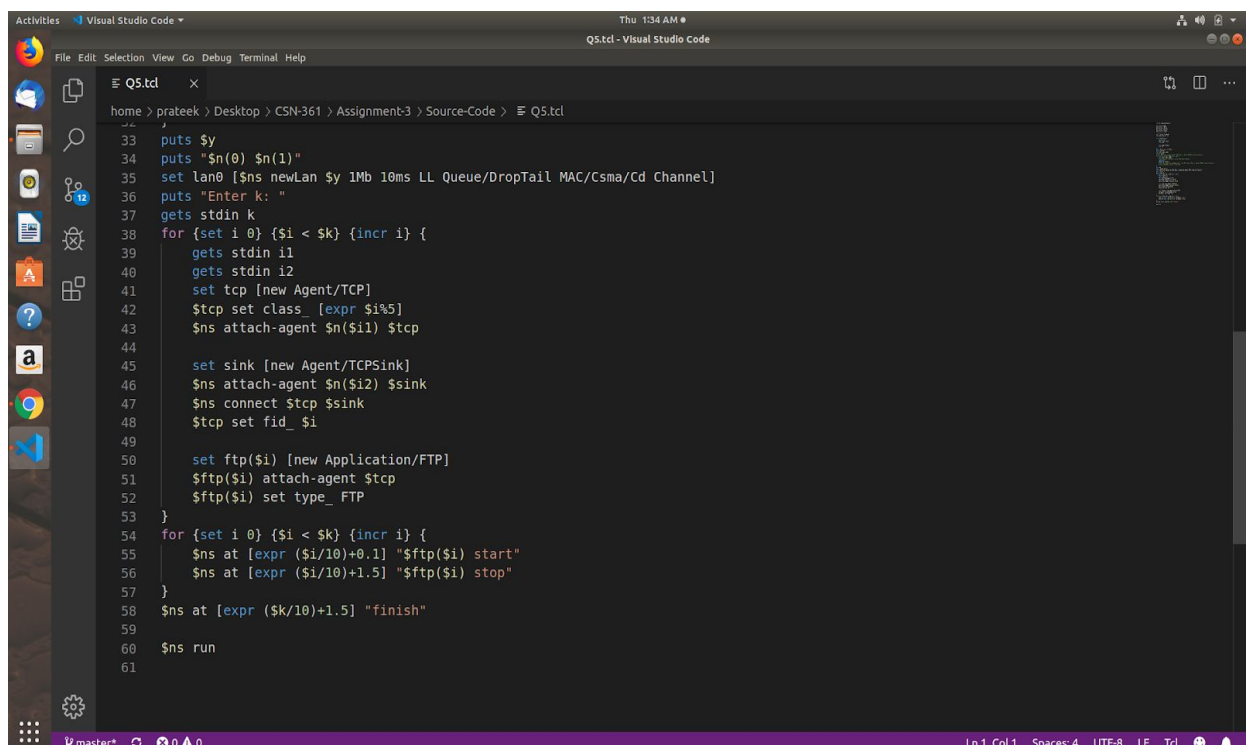
## Implementations details

We have implemented bus topology. A bus topology is a topology for a Local Area Network (LAN) in which all the nodes are connected to a single cable.

## Code Snippets



```
1 set ns [new Simulator]
2
3 $ns color 0 Red
4 $ns color 1 Blue
5 $ns color 2 Azure
6 $ns color 3 Coral
7 $ns color 4 Cyan
8
9 set f [open 5.nam w]
10 $ns namtrace-all $f
11
12 proc finish {} {
13     global ns f
14     $ns flush-trace
15     close $f
16
17     exec nam 5.nam &
18     exit 0
19 }
20 puts "Enter no. of Nodes: "
21 gets stdin N
22 set n(0) [$ns node]
23 set y "$n(0)"
24 # set lan0 [$ns newLan "$n(0)" 1Mb 10ms LL Queue/FQ MAC/Csma/Cd Channel]
25 for {set i 1} {$i < $N} {incr i} {
26     set n($i) [$ns node]
27     # $ns duplex-link $n($i) $n(0) 1Mb 10ms DropTail
28     append y " "
29     append y "$n($i)"
30     # set lan [eval new LanNode $self -bw 1Mb -delay 10ms LL Queue/FQ MAC/Csma/Cd Channel]
31     # $lan0 addNode "$n($i)" 1Mb 10ms
32 }
```



```
33 puts $y
34 puts "$n(0) $n(1)"
35 set lan0 [$ns newLan $y 1Mb 10ms LL Queue/DropTail MAC/Csma/Cd Channel]
36 puts "Enter k: "
37 gets stdin k
38 for {set i 0} {$i < $k} {incr i} {
39     gets stdin i1
40     gets stdin i2
41     set tcp [new Agent/TCP]
42     $tcp set class_ [expr $i%5]
43     $ns attach-agent $n($i1) $tcp
44
45     set sink [new Agent/TCPSink]
46     $ns attach-agent $n($i2) $sink
47     $ns connect $tcp $sink
48     $tcp set fid_ $i
49
50     set ftp($i) [new Application/FTP]
51     $ftp($i) attach-agent $tcp
52     $ftp($i) set type_FTP
53 }
54 for {set i 0} {$i < $k} {incr i} {
55     $ns at [expr ($i/10)+0.1] "$ftp($i) start"
56     $ns at [expr ($i/10)+1.5] "$ftp($i) stop"
57 }
58 $ns at [expr ($k/10)+1.5] "finish"
59
60 $ns run
61 }
```



## Snapshots of running code

