Computer Network
Practical Assignment

Submitted by:-
Jyotiswaroop Srivastav
21/18023
B.Sc.(Hons.) Comp Sci.

Submitted to:-
Ms. Uma Ojha
Department of Computer Science

Practical 1

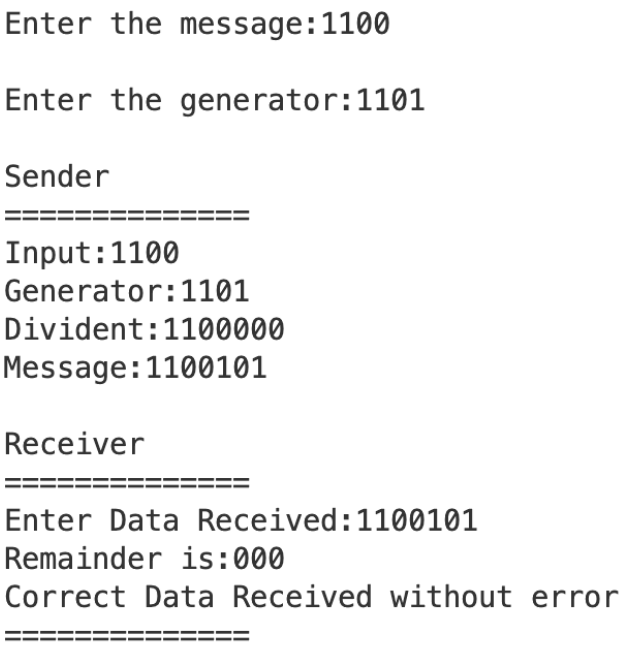
Objective:-

Simulate Cyclic Redundancy Check (CRC) error detection algorithm for a noisy channel.

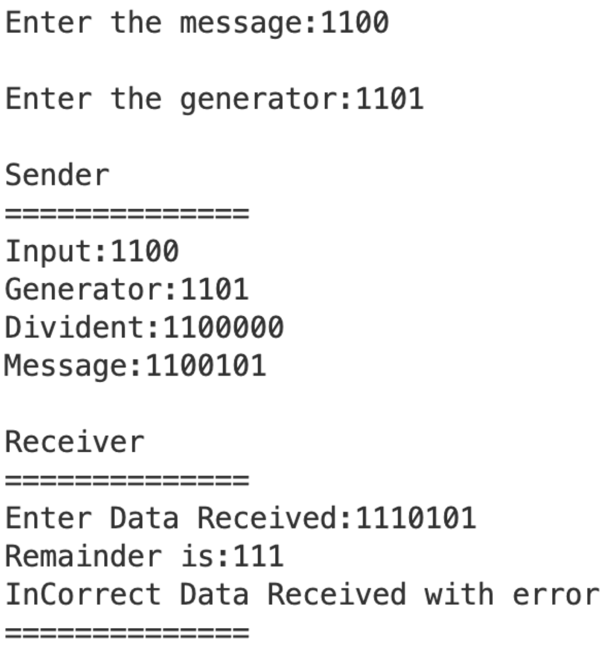
Code:-

```
#include <iostream>
#include <string.h>
using namespace std;
class CRC
{
    string input, divisor, divident, result;
    int len_divident, len_gen, len_inp;

public:
    string fun_xor(string a, string b)
    {
        string res = "";
        if (a[0] == '0')
        {
            return a.substr(1);
        }
        else
        {
            for (int i = 0; i < len_gen; i++)
            {
                res = res + (a[i] == b[i] ? '0' : '1');
            }
            return res.substr(1);
        }
    }
    void modulo_div()
    {
        string temp_div = divisor;
        string temp_divident = divident.substr(0, len_gen);
        int j = len_gen;
        while (j < len_divident)
        {
            temp_divident = fun_xor(temp_divident, temp_div);
            temp_divident = temp_divident + divident[j];
            j++;
        }
        result = input + fun_xor(temp_divident, temp_div);
    }
    void getdata()
    {
        cout << "\nEnter the message:";
        cin >> input;
        cout << "\nEnter the generator:";
        cin >> divisor;
        len_gen = divisor.length();
        len_inp = input.length();
        divident = input;
        for (int i = 0; i < len_gen - 1; i++)
        {
            divident += '0';
        }
        len_divident = divident.length();
        modulo_div();
    }
    void sender_side()
    {
        cout << "\nSender\n==============\n";
        cout << "Input:" << input << endl;
        cout << "Generator:" << divisor << endl;
        cout << "Divident:" << divident << endl;
        cout << "Message:" << result << endl;
    }
    void receiver_side()
    {
        string data_rec;
        cout << "\nReceiver\n==============\n";
        cout << "Enter Data Received:";
        cin >> data_rec;

        string temp_div = divisor;
        string temp_divident = data_rec.substr(0, len_gen);
        int j = len_gen;
        while (j < data_rec.length())
        {
            temp_divident = fun_xor(temp_divident, temp_div);
            temp_divident =temp_divident + data_rec[j];
            j++;
        }
        string error = fun_xor(temp_divident, temp_div);
        cout << "Remainder is:" << error << endl;

        bool flag = 0;
        for (int i = 0; i < len_gen - 1; i++)
        {
            if (error[i] == '1')
            {
                flag = 1;
                break;
            }
        }
        if (flag)
        {
            cout << "InCorrect Data Received with error" << endl;
        }
        else
        {
            cout << "Correct Data Received without error" << endl;
        }
        cout << "=============\n";
    }
};
int main()
{
    CRC crc;
    crc.getdata();
    crc.sender_side();
    crc.receiver_side();
    return 0;
}
```

Output:-



```
Enter the message:1100

Enter the generator:1101

Sender
==============
Input:1100
Generator:1101
Divident:1100000
Message:1100101

Receiver
==============
Enter Data Received:1100101
Remainder is:000
Correct Data Received without error
==============
```



```
Enter the message:1100

Enter the generator:1101

Sender
==============
Input:1100
Generator:1101
Divident:1100000
Message:1100101

Receiver
==============
Enter Data Received:1110101
Remainder is:111
InCorrect Data Received with error
==============
```

Practical 2

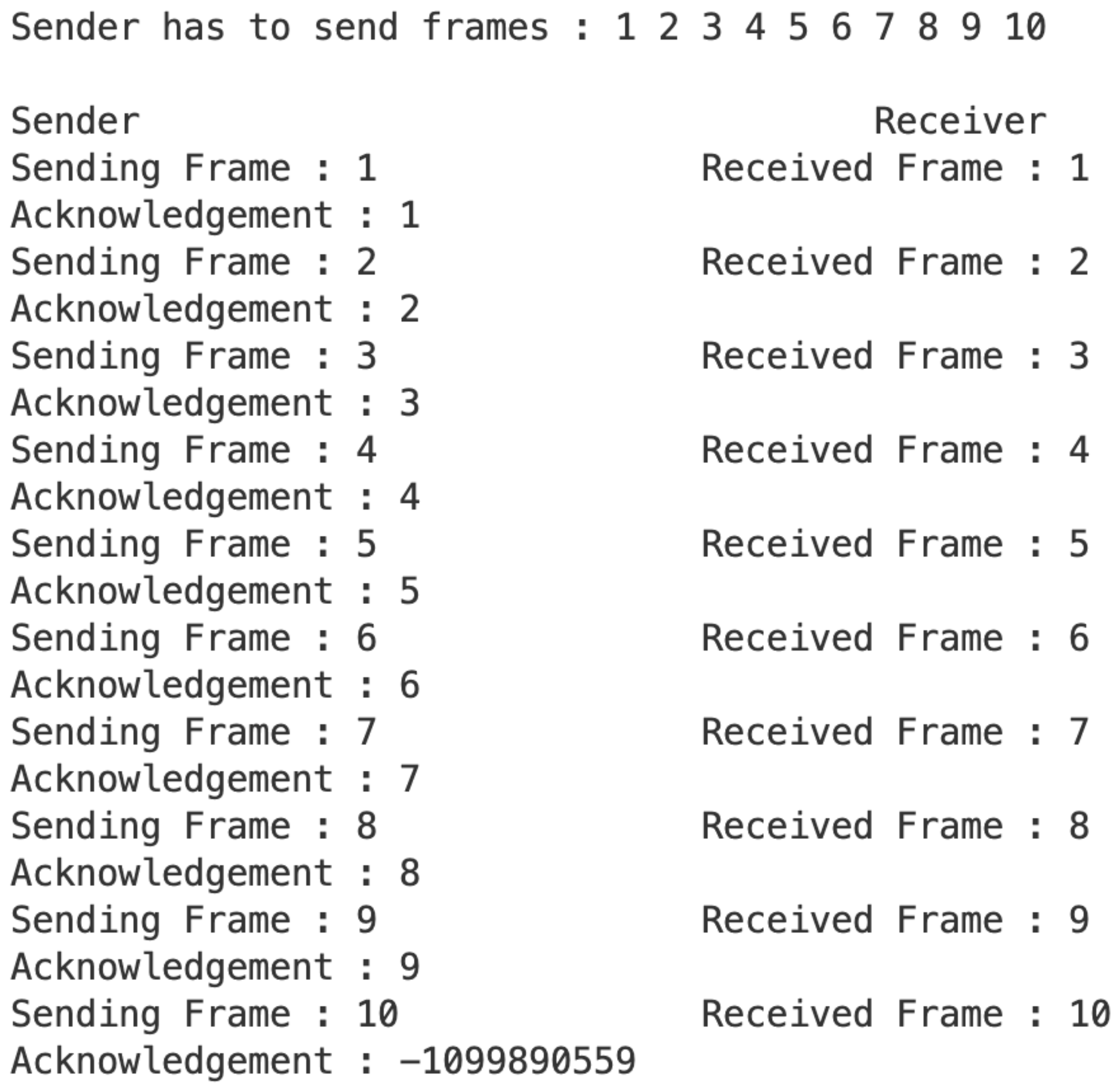
Objective:-

Simulate and implement stop and wait protocol for noisy channel.

Code:-

```
#include <iostream>
#include <time.h>
#include <cstdlib>
#include <ctime>
#include <unistd.h>
using namespace std;
class timer
{
private:
    unsigned long begTime;

public:
    void start()
    {
        begTime = clock();
    }
    unsigned long elapsedTime()
    {
        return ((unsigned long)clock() - begTime) / CLOCKS_PER_SEC;
    }
    bool isTimeout(unsigned long seconds)
    {
        return seconds >= elapsedTime();
    }
};
int main()
{
    int frames[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    unsigned long seconds = 5;
    srand(time(NULL));
    timer t;
    cout << "Sender has to send frames : ";
    for (int i = 0; i < 10; i++)
        cout << frames[i] << " ";
    cout << endl;
    int count = 0;
    bool delay = false;
    cout << endl
         << "Sender\t\t\t\tReceiver" << endl;
    do
    {
        bool timeout = false;
        cout << "Sending Frame : " << frames[count];
        cout.flush();
        cout << "\t\t";
        t.start();
        if (rand() % 2)
        {
            int to = 24600 + rand() % (64000 - 24600) + 1;
            for (int i = 0; i < 64000; i++)
                for (int j = 0; j < to; j++)
                {
                }
        }
        if (t.elapsedTime() <= seconds)
        {
            cout << "Received Frame : " << frames[count] << " ";
            if (delay)
            {
                cout << "Duplicate";
                delay = false;
            }
            cout << endl;
            count++;
        }
        else
        {
            cout << "---" << endl;
            cout << "Timeout" << endl;
            timeout = true;
        }
        t.start();
        if (rand() % 2 || !timeout)
        {
            int to = 24600 + rand() % (64000 - 24600) + 1;
            for (int i = 0; i < 64000; i++)
                for (int j = 0; j < to; j++)
                {
                }
            if (t.elapsedTime() > seconds)
            {
                cout << "Delayed Ack" << endl;
                count--;
                delay = true;
            }
            else if (!timeout)
                cout << "Acknowledgement : " << frames[count] - 1 << endl;
        }
    } while (count != 10);
    return 0;
}
```

Output:-



```
Sender has to send frames : 1 2 3 4 5 6 7 8 9 10

Sender                                  Receiver
Sending Frame : 1               Received Frame : 1
Acknowledgement : 1
Sending Frame : 2               Received Frame : 2
Acknowledgement : 2
Sending Frame : 3               Received Frame : 3
Acknowledgement : 3
Sending Frame : 4               Received Frame : 4
Acknowledgement : 4
Sending Frame : 5               Received Frame : 5
Acknowledgement : 5
Sending Frame : 6               Received Frame : 6
Acknowledgement : 6
Sending Frame : 7               Received Frame : 7
Acknowledgement : 7
Sending Frame : 8               Received Frame : 8
Acknowledgement : 8
Sending Frame : 9               Received Frame : 9
Acknowledgement : 9
Sending Frame : 10              Received Frame : 10
Acknowledgement : -1099890559
```

Practical 3

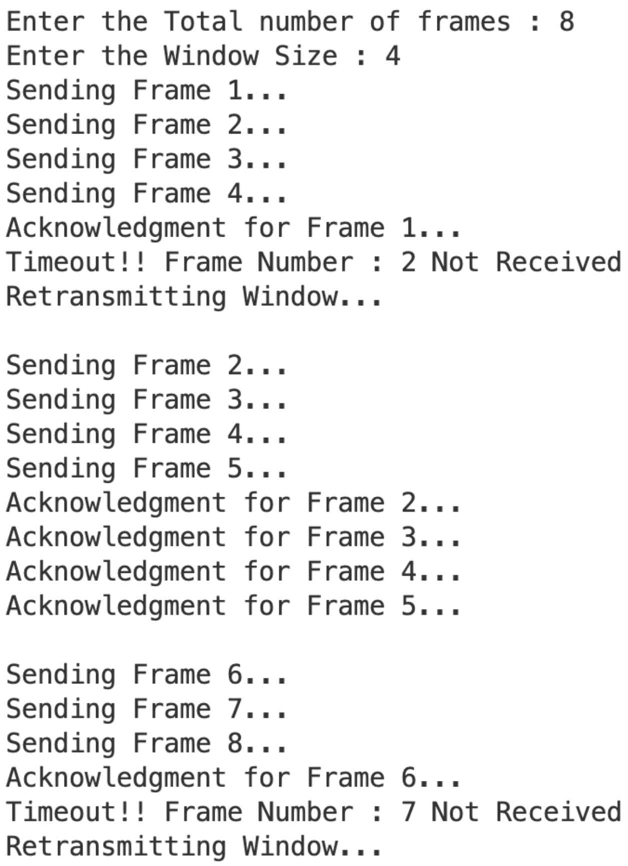
Objective:-

Simulate and implement Go - Back sliding window protocol.

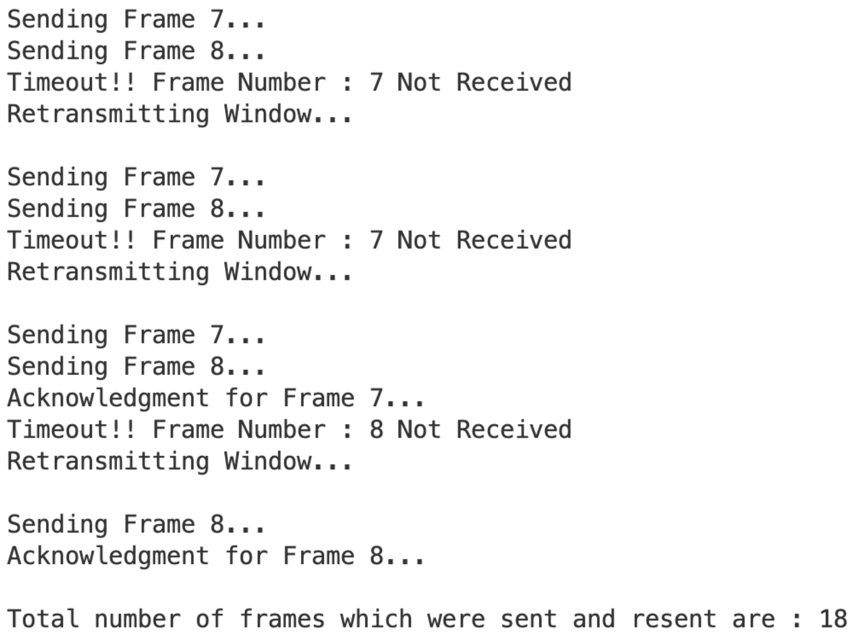
Code:-

```
#include<iostream>
#include<ctime>
#define ll long long int
using namespace std;

void transmission(ll & i, ll & N, ll & tf, ll & tt) {
  while (i <= tf) {
    int z = 0;
    for (int k = i; k < i + N && k <= tf; k++) {
      cout << "Sending Frame " << k << "..." << endl;
      tt++;
    }
    for (int k = i; k < i + N && k <= tf; k++) {
      int f = rand() % 2;
      if (!f) {
        cout << "Acknowledgment for Frame " << k << "..." << endl;
        z++;
      } else {
        cout << "Timeout!! Frame Number : " << k << " Not Received" << endl;
        cout << "Retransmitting Window..." << endl;
        break;
      }
    }
    cout << "\n";
    i = i + z;
  }
}

int main() {
  ll tf, N, tt = 0;
  srand(time(NULL));
  cout << "Enter the Total number of frames : ";
  cin >> tf;
  cout << "Enter the Window Size : ";
  cin >> N;
  ll i = 1;
  transmission(i, N, tf, tt);
  cout << "Total number of frames which were sent and resent are : " << tt <<
    endl;
  return 0;
}
```

Output:-



```
Enter the Total number of frames : 8
Enter the Window Size : 4
Sending Frame 1...
Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Acknowledgment for Frame 1...
Timeout!! Frame Number : 2 Not Received
Retransmitting Window...

Sending Frame 2...
Sending Frame 3...
Sending Frame 4...
Sending Frame 5...
Acknowledgment for Frame 2...
Acknowledgment for Frame 3...
Acknowledgment for Frame 4...
Acknowledgment for Frame 5...

Sending Frame 6...
Sending Frame 7...
Sending Frame 8...
Acknowledgment for Frame 6...
Timeout!! Frame Number : 7 Not Received
Retransmitting Window...
```



```
Sending Frame 7...
Sending Frame 8...
Timeout!! Frame Number : 7 Not Received
Retransmitting Window...

Sending Frame 7...
Sending Frame 8...
Timeout!! Frame Number : 7 Not Received
Retransmitting Window...

Sending Frame 7...
Sending Frame 8...
Acknowledgment for Frame 7...
Timeout!! Frame Number : 8 Not Received
Retransmitting Window...

Sending Frame 8...
Acknowledgment for Frame 8...

Total number of frames which were sent and resent are : 18
```

Practical 4

Objective:-

Simulate and implement selective repeat sliding window protocol.

Code:-

//protocol.hpp

```
#include <cstdio>
#include <string>
#include <iostream>
#define MAX_PKT 5
using namespace std;
typedef enum
{
  dat,
  ack,
  nak
} frameKind;
typedef enum
{
  timeout,
  checksumError,
  frameArrival,
  networkLayerReady
} eventType;
typedef struct
{
  unsigned char data;
} packet;
typedef struct
{
  packet *info;
  frameKind kind;
  unsigned int seq;
  unsigned int ack;
} frame;
class Protocol
{
public:
  eventType event;
  bool noNak, errorDetected;
  int MAX_SEQ, flag, err, buf;
  int frameExpected, frameToSend;
  packet dataPacket;
  frame senderFrame, receiverFrame;
  Protocol()
  {
    noNak = true;
    err = buf = -1;
    flag = frameToSend = frameExpected = 0;
    errorDetected = false;
  }
  int waitForEvent(eventType e)
  {
    return e == frameArrival;
  }
  string showkind(frameKind k)
  {
    switch (k)
    {
    case dat:
      return "data";
      break;
    case ack:
      return "ack";
      break;
    case nak:
      return "nak";
      break;
    }
    return "";
  }
  // Network -> Data Link Interface
  void fromNetworkLayer(packet &i)
  {
    printf("\nEncapsulating Packet<data='%c'> ...", i.data);
    senderFrame.seq = frameToSend;
    senderFrame.kind = dat;
    senderFrame.info = &i;
    frameToSend = (frameToSend + 1) % (MAX_SEQ + 1);
  }
  // Data Link -> Physical Interface
  void toPhysicalLayer(frame &f)
  {
    if (event == timeout)
    {
      cout << "\nTimeout period expired. Resending frame with sequence no. " << err;
      f.seq = err;
      err = -1;
      frameToSend = (err + 1) % (MAX_SEQ + 1);
      event = frameArrival;
    }
    else if (f.kind == dat)
    printf("\nSending DataFrame<kind=%s, sequence=%i> to Physical Layer
...",showkind(f.kind).c_str(),f.seq);
    else
    {
```

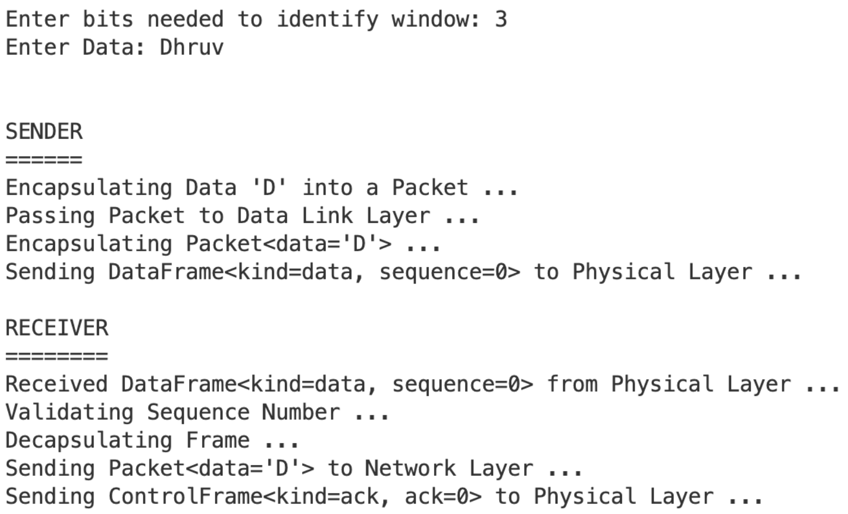
```
   if (err != -1)
   {
    if (!noNak)
    {
     f.kind = nak;
     f.ack = err;
     noNak = true;
    }
    else
    {
     f.kind = ack;
     f.ack = err - 1;
    }
   }
   else if (buf != -1)
   {
    f.ack = buf;
    frameExpected = (buf + 1) % (MAX_SEQ + 1);
    frameToSend = frameExpected;
    buf = -1;
 }
 printf("\nSending ControlFrame<kind=%s, ack=%i> to Physical Layer
...",showkind(f.kind).c_str(), f.ack);
 }
 }
 // Data Link -> Network Interface
 void toNetworkLayer(packet &p)
 {
  printf("\nSending Packet<data='%c'> to Network Layer ...", p.data);
  receiverFrame.seq = frameToSend - 1;
  receiverFrame.kind = ack;
  receiverFrame.ack = frameExpected;
  frameExpected = (frameExpected + 1) % (MAX_SEQ + 1);
 }
 // Physical -> Data Link Interface
 void fromPhysicalLayer(frame &f)
 {
  printf("\nReceived DataFrame<kind=%s, sequence=%i> from Physical Layer
...",showkind(f.kind).c_str(),f.seq);
  printf("\nValidating Sequence Number ... ");
  {
   if (frameExpected == f.seq)
   {
    if (f.seq == 1 && flag == 0) // Error Simulation
    {
     cout << "\nError in received frame ...";
    flag = 1;
    noNak = false;
    errorDetected = true;
    err = f.seq;
   }
   else
   {
    printf("\nDecapsulating Frame ...");
    noNak = true;
    toNetworkLayer(dataPacket);
   }
  }
  else
  {
   printf("\nFrame out of order. Storing in buffer ...");
   buf = f.seq;
  }
 }
}
}
;
```

//main.cpp

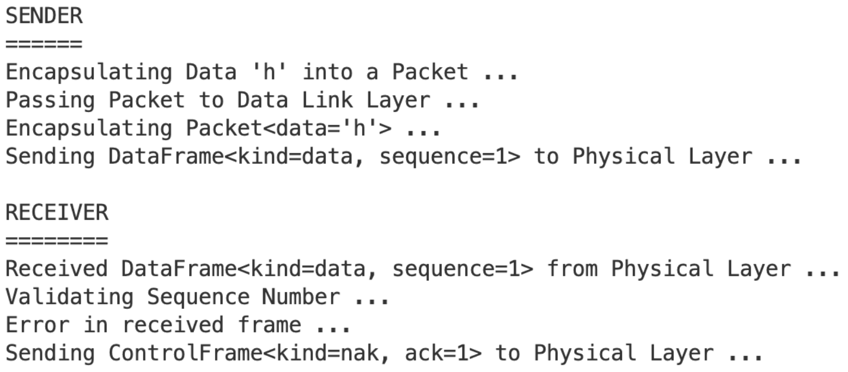
```
#include <cstring>
#include <cstdlib>
#include <cmath>
#include "protocol.hpp"
using namespace std;
void getch()
{
  cin.ignore();
  cin.get();
  return;
}
void clrscr()
{
#ifdef _WIN32
  system("cls");
#elif __unix__
  system("clear");
#endif
  return;
}
class selectiveRepeatSlidingWindow : public Protocol
{
public:
  string in_buf;
  selectiveRepeatSlidingWindow(int n, string s)
  {
    MAX_SEQ = n;
    in_buf = s;
  }
```

```
    void sender();
    void receiver();
};
void selectiveRepeatSlidingWindow::sender()
{
    event = frameArrival;
    printf("\n\nSENDER\n======");
    if (frameToSend ==
            (err + (MAX_SEQ / 2)) %
                (MAX_SEQ + 1) &&
        errorDetected == true &&
        err >= 0)
    {
        event = timeout;
        frameToSend = err;
        errorDetected = 0;
    }
    else if (frameToSend == MAX_SEQ && frameToSend != frameExpected &&
errorDetected == true)
    {
        fromNetworkLayer(dataPacket);
        frameToSend = frameExpected;
        errorDetected = false;
    }
    else if (event == frameArrival)
    {
        printf("\nEncapsulating Data '%c' into a Packet ...", in_buf[frameToSend]);
        dataPacket.data = in_buf[frameToSend];
        printf("\nPassing Packet to Data Link Layer ...");
        fromNetworkLayer(dataPacket);
    }
    toPhysicalLayer(senderFrame);
    receiver();
}
void selectiveRepeatSlidingWindow::receiver()
{
    printf("\n\nRECEIVER\n========");
    fromPhysicalLayer(senderFrame);
    toPhysicalLayer(receiverFrame);
    getch();
    clrscr();
    sender();
}
int main()
{
    int n;
    cout << "\nEnter bits needed to identify window: ";
    cin >> n;
    char temp[50];
    printf("Enter Data: ");
    scanf("%s", temp);
    selectiveRepeatSlidingWindow *obj = new selectiveRepeatSlidingWindow(
        pow(2, n) - 1,
        string(temp));
    obj->sender();
    delete obj;
    return 0;
}
```

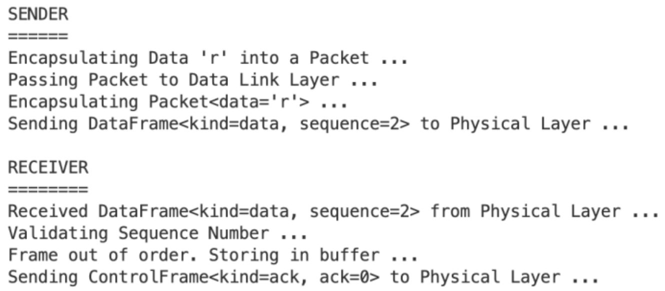
Output:-



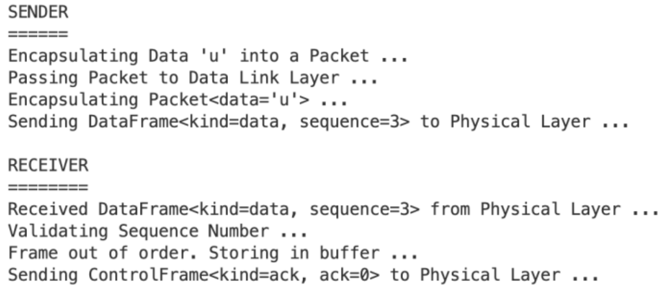
```
Enter bits needed to identify window: 3
Enter Data: Dhruv


SENDER
======
Encapsulating Data 'D' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='D'> ...
Sending DataFrame<kind=data, sequence=0> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=0> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='D'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=0> to Physical Layer ...
```



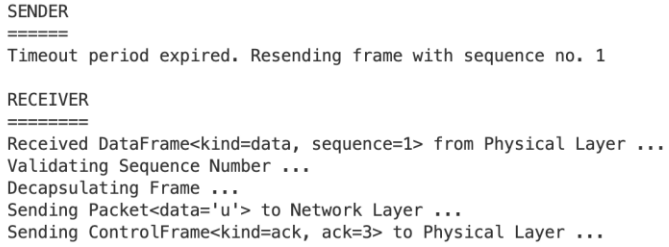
```
SENDER
======
Encapsulating Data 'h' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='h'> ...
Sending DataFrame<kind=data, sequence=1> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=1> from Physical Layer ...
Validating Sequence Number ...
Error in received frame ...
Sending ControlFrame<kind=nak, ack=1> to Physical Layer ...
```



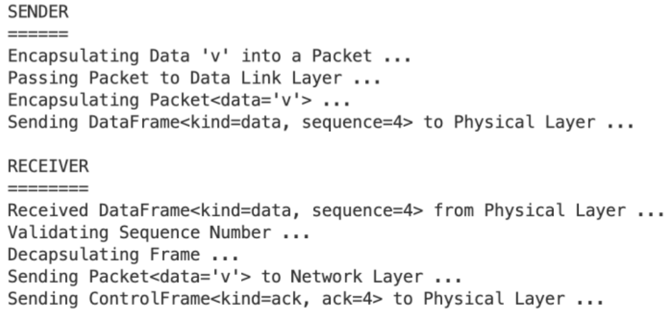
```
SENDER
======
Encapsulating Data 'r' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='r'> ...
Sending DataFrame<kind=data, sequence=2> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=2> from Physical Layer ...
Validating Sequence Number ...
Frame out of order. Storing in buffer ...
Sending ControlFrame<kind=ack, ack=0> to Physical Layer ...
```



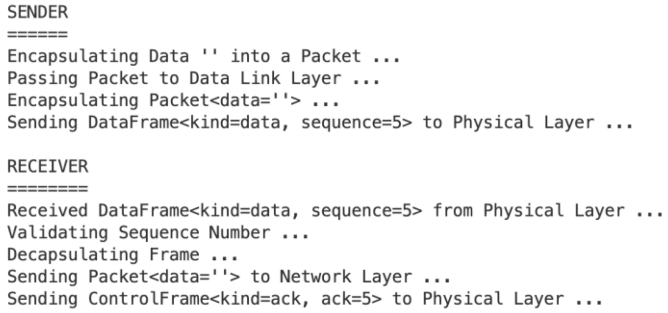
```
SENDER
======
Encapsulating Data 'u' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='u'> ...
Sending DataFrame<kind=data, sequence=3> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=3> from Physical Layer ...
Validating Sequence Number ...
Frame out of order. Storing in buffer ...
Sending ControlFrame<kind=ack, ack=0> to Physical Layer ...
```



```
SENDER
======
Timeout period expired. Resending frame with sequence no. 1

RECEIVER
========
Received DataFrame<kind=data, sequence=1> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='u'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=3> to Physical Layer ...
```



```
SENDER
======
Encapsulating Data 'v' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data='v'> ...
Sending DataFrame<kind=data, sequence=4> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=4> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data='v'> to Network Layer ...
Sending ControlFrame<kind=ack, ack=4> to Physical Layer ...
```



```
SENDER
======
Encapsulating Data '' into a Packet ...
Passing Packet to Data Link Layer ...
Encapsulating Packet<data=''> ...
Sending DataFrame<kind=data, sequence=5> to Physical Layer ...

RECEIVER
========
Received DataFrame<kind=data, sequence=5> from Physical Layer ...
Validating Sequence Number ...
Decapsulating Frame ...
Sending Packet<data=''> to Network Layer ...
Sending ControlFrame<kind=ack, ack=5> to Physical Layer ...
```

Practical 5

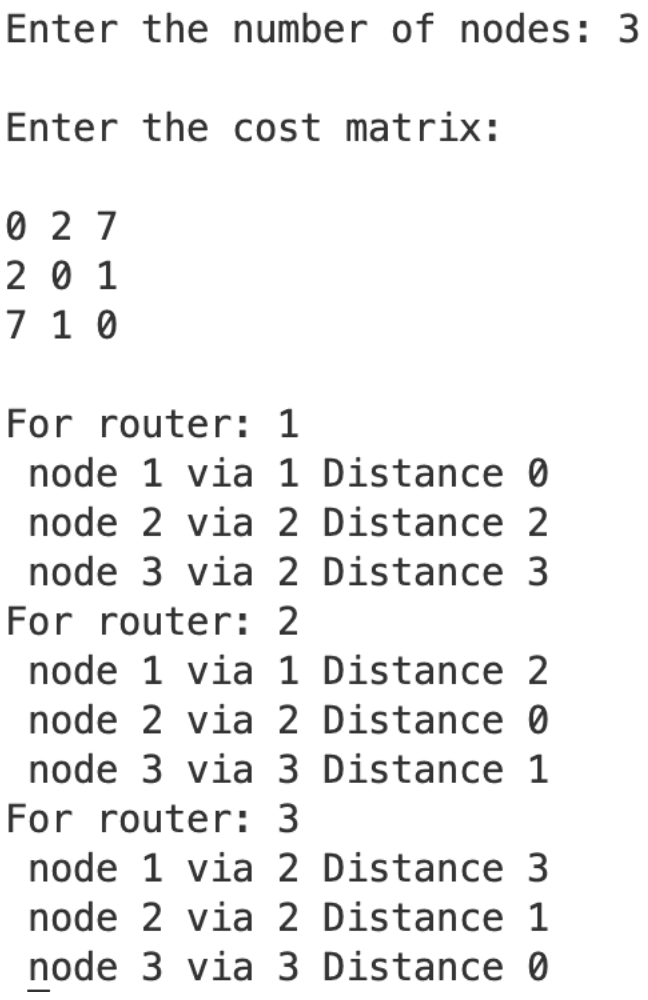
Objective:-

Simulate and implement distance vector routing algorithm

Code:-

```
#include <iostream>
using namespace std;
struct node
{
    unsigned dist[20];
    unsigned from[20];
} dvr[10];
int main()
{
    int cost[20][20];
    int i, j, k, nodes, count = 0;
    cout << "\nEnter the number of nodes: ";
    cin >> nodes;
    cout << "\nEnter the cost matrix: \n";
    for (i = 0; i < nodes; i++)
    {
        for (j = 0; j < nodes; j++)
        {
            cin >> cost[i][j];
            cost[i][i] = 0;
            dvr[i].dist[j] = cost[i][j]; // initializing distance equal to cost matrix
            dvr[i].from[j] = j;
        }
    }
    do
    {
        count = 0;
        for (i = 0; i < nodes; i++)
            for (j = 0; j < nodes; j++)
                for (k = 0; k < nodes; k++)
                    if (dvr[i].dist[j] > cost[i][k] + dvr[k].dist[j])
                    { // calculate the minimum distance
                        dvr[i].dist[j] = dvr[i].dist[k] + dvr[k].dist[j];
                        dvr[i].from[j] = k;
                        count++;
                    }
    } while (count != 0);
    for (i = 0; i < nodes; i++)
    {
        cout << "\nFor router: " << i + 1;
        for (j = 0; j < nodes; j++)
        {
            cout << "\t\n node " << j + 1 << " via " << dvr[i].from[j] + 1 << " Distance " <<
dvr[i].dist[j];
        }
    }
    cout << endl;
}
```

Output:-



```
Enter the number of nodes: 3

Enter the cost matrix:

0 2 7
2 0 1
7 1 0

For router: 1
 node 1 via 1 Distance 0
 node 2 via 2 Distance 2
 node 3 via 2 Distance 3
For router: 2
 node 1 via 1 Distance 2
 node 2 via 2 Distance 0
 node 3 via 3 Distance 1
For router: 3
 node 1 via 2 Distance 3
 node 2 via 2 Distance 1
 node 3 via 3 Distance 0
```

Practical 6

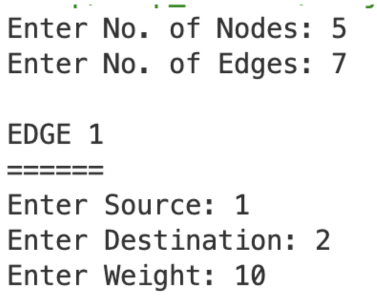
Objective:-

Simulate and implement Dijkstra algorithm for shortest path routing.

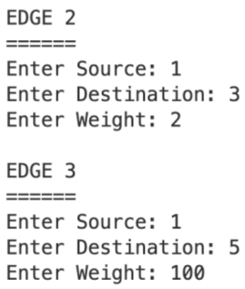
Code:-

```cpp
#include <cstdio>
#include <climits>
#include <iomanip>
#include <iostream>
#define MAX_NODES 10
using namespace std;
class Graph
{
public:
    int edges;
    int vertices;
    int path[MAX_NODES];
    int distances[MAX_NODES];
    int adjMatrix[MAX_NODES][MAX_NODES];
    void input(int v, int e)
    {
        edges = e;
        vertices = v;
        // initialize the adjacency matrix
        for (int i = 0; i < v; i++)
            for (int j = 0; j < v; j++)
                adjMatrix[i][j] = 0;
        int src, dest, weight;
        // populate the adjacency matrix
        for (int i = 0; i < edges; i++)
        {
            cout << "\nEDGE " << (i + 1)
                 << "\n======\n";
            cout << "Enter Source: ";
            cin >> src;
            cout << "Enter Destination: ";
            cin >> dest;
            cout << "Enter Weight: ";
            cin >> weight;
            adjMatrix[src - 1][dest - 1] = weight;
            adjMatrix[dest - 1][src - 1] = weight;
        }
    }
    void display()
    {
        for (int i = 0; i < vertices; i++)
        {
            for (int j = 0; j < vertices; j++)
```

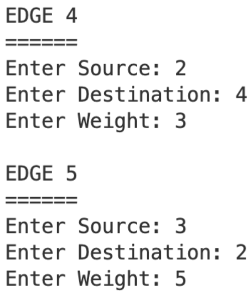
```
                cout << setw(5) << adjMatrix[i][j] << " ";
            cout << endl;
        }
    }
    void dijkstra(int src)
    {
        bool visited[MAX_NODES];
        for (int i = 0; i < vertices; i++)
        {
            visited[i] = false;     // mark node as not processed
            distances[i] = INT_MAX; // set distance from src as infinity
        }
        // mark the src node
        path[src] = -1;
        distances[src] = 0;
        // iterate over all vertices
        for (int i = 0; i < vertices - 1; i++)
        {
            // find the nearest unprocessed node
            int u = minDistance(visited); // mark node as processed visited[u] = true;
            // iterate over all nodes
            for (int v = 0; v < vertices; v++)
                // update distance for unprocessed node if there // exists an edge(u,v) and new
distance is lesser // also add the node to the shortest path
                if (visited[v] == false && adjMatrix[u][v] && distances[u] != INT_MAX &&
distances[u] + adjMatrix[u][v] < distances[v])
                {
                    path[v] = u;
                    distances[v] = distances[u] + adjMatrix[u][v];
                }
        }
        // print distances and shortest paths
        cout << "\nDest Node \t Distance \t Shortest Path";
        cout << "\n========= \t ======== \t =============";
        for (int i = 0; i < vertices; i++)
        {
            cout << endl
                 << (i + 1)
                 << " \t\t " << distances[i]
                 << " \t\t " << (src + 1);
            printShortestPath(i);
        }
    }
    int minDistance(bool *visited)
    {
        int min = INT_MAX, min_index;
        for (int v = 0; v < vertices; v++)
            if (visited[v] == false && distances[v] <= min)
            {
                min = distances[v];
                min_index = v;
            }
        return min_index;
    }
    void printShortestPath(int node)
    {
        if (path[node] == -1)
            return;
        printShortestPath(path[node]);
        cout << " -> " << (node + 1);
    }
};
int main()
{
    int v, e;
    Graph graph;
    cout << "Enter No. of Nodes: ";
    cin >> v;
    cout << "Enter No. of Edges: ";
    cin >> e;
    graph.input(v, e);
    cout << "\nGRAPH\n=====\n";
    graph.display();
    cout << endl;
    cout << "Enter Source Node: ";
    cin >> v;
    graph.dijkstra(v - 1);
    return 0;
}
```



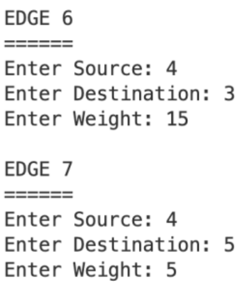
```
Enter No. of Nodes: 5
Enter No. of Edges: 7

EDGE 1
======
Enter Source: 1
Enter Destination: 2
Enter Weight: 10
```



```
EDGE 2
======
Enter Source: 1
Enter Destination: 3
Enter Weight: 2

EDGE 3
======
Enter Source: 1
Enter Destination: 5
Enter Weight: 100
```

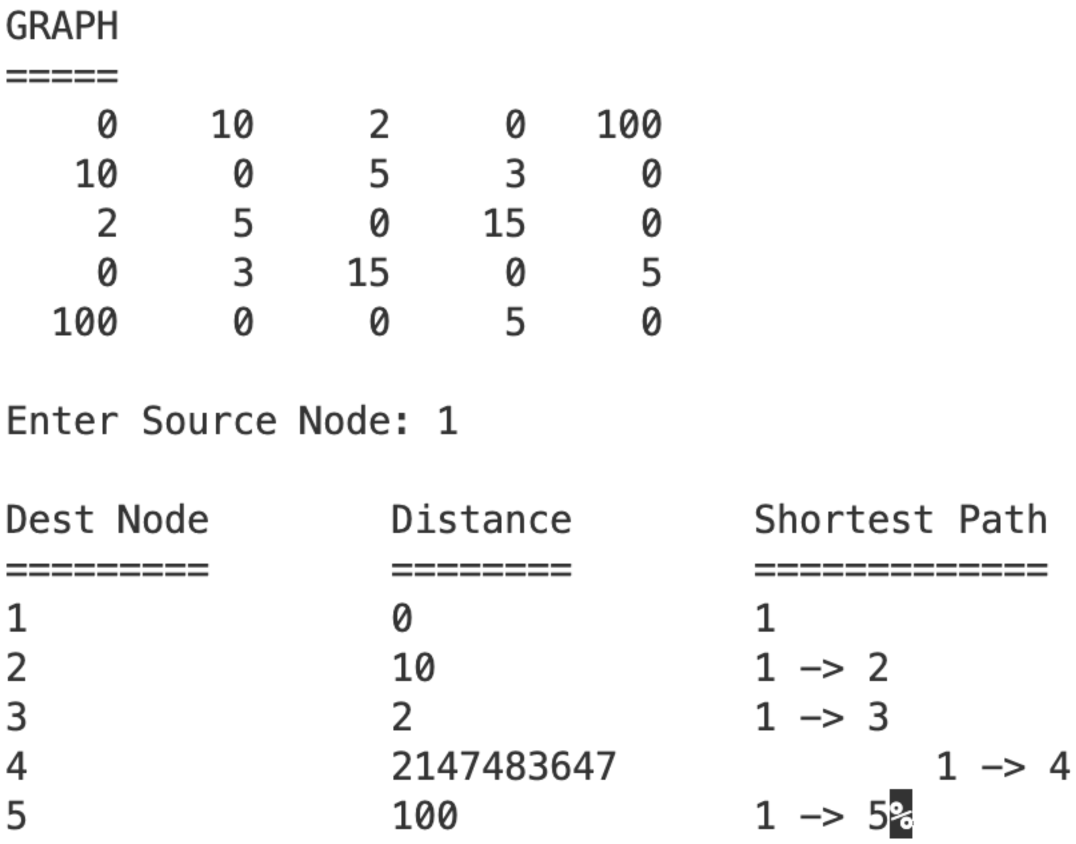


```
EDGE 4
======
Enter Source: 2
Enter Destination: 4
Enter Weight: 3

EDGE 5
======
Enter Source: 3
Enter Destination: 2
Enter Weight: 5
```



```
EDGE 6
======
Enter Source: 4
Enter Destination: 3
Enter Weight: 15

EDGE 7
======
Enter Source: 4
Enter Destination: 5
Enter Weight: 5
```

Output:-



```
GRAPH
=====
     0    10     2     0   100
    10     0     5     3     0
     2     5     0    15     0
     0     3    15     0     5
   100     0     0     5     0

Enter Source Node: 1

Dest Node                Distance                Shortest Path
=========                ========                =============
1                        0                       1
2                        10                      1 -> 2
3                        2                       1 -> 3
4                        2147483647                      1 -> 4
5                        100                     1 -> 5%
```