INDEX

Program 1:

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

```
#include <iostream>
using namespace std;
int xorp(int a, int b)
{
   if (a != b)
   {
      return 1;
   }
   return 0;
}
void crc1(int size, int size2, int data[], int divide[], int crc[], int polygen[])
{
   int s = size;
   int it = size;
   while (s <= size2)
   {

      if (divide[0] == 1)
      {

         for (int i = 0; i < size; i++)
         {
            int a = xorp(divide[i], polygen[i]);
```

```
            if (i != 0)
            {
                crc[i - 1] = a;
            }
        }
    }
    else
    {
        for (int i = 0; i < size; i++)
        {
            int a = xorp(divide[i], 0);
            if (i != 0)
            {
                crc[i - 1] = a;
            }
        }
    }

    for (int i = 0; i < size - 1; i++)
    {
        divide[i] = crc[i];
    }
    divide[size - 1] = data[it];
    it++;
    s++;
  }
}
```

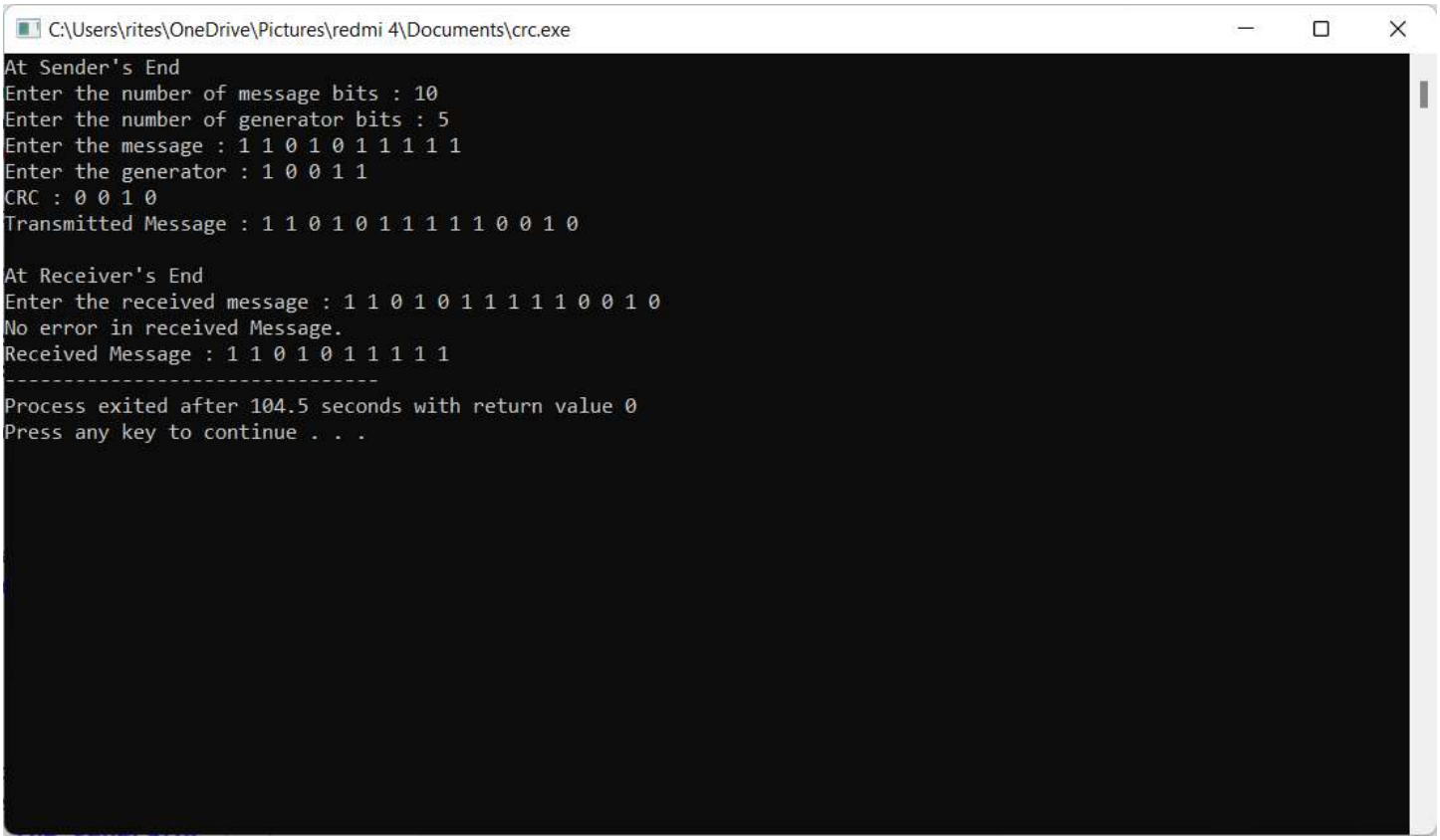
```
int main()
{
   int size, size1;
   cout << "enter the size of generator: ";
   cin >> size;
   cout << "Enter the size of data: ";
   cin >> size1;
   int polygen[size];
   int size2 = size1 + (size - 1);
   int data[size2];
   cout << "enter the polynomial generator in form of (0 1) \n";

   for (int i = 0; i < size; i++)
   {
      cin >> polygen[i];
   }
   cout << "Enter the data in form of(0 1)\n";
   for (int i = 0; i < size1; i++)
   {
      cin >> data[i];
   }
   for (int i = size1; i < size2; i++)
   {
      data[i] = 0;
   }

   cout << "The polynomial generator are: ";
```

```
for (int i = 0; i < size; i++)
{
   cout << polygen[i] << " ";
}
cout << "\ndata include (n-1) 0 added;  ";
for (int i = 0; i < size2; i++)
{
   cout << data[i] << " ";
}

int divide[size];
for (int i = 0; i < size; i++)
{
   divide[i] = data[i];
}
int crc[size - 1];
crc1(size, size2, data, divide, crc, polygen);

cout << "\n crc is:  ";
for (int i = 0; i < size - 1; i++)
{
   cout << crc[i] << " ";
}

for (int i = 0; i < size - 1; i++)
{
   data[size1 + i] = crc[i];
```

```
}
cout << "\n the receive data is : ";
for (int i = 0; i < size2; i++)
{
   cout << data[i] << " ";
}
cout << endl;
for (int i = 0; i < size; i++)
{
   divide[i] = data[i];
}
crc1(size, size2, data, divide, crc, polygen);
cout << "\n crc is:  ";
int count=0;
for (int i = 0; i < size - 1; i++)
{
   if (crc[i] == 0)
   {
      count = count + 1;
   }
   cout << crc[i] << " ";
}
if (count == size - 1)
{
   cout << "\nNo error detected ";
}
else
```

```
    {
        cout << "\nerror detected here data is not correct";
    }
    return 0;
}
```

OUTPUT:

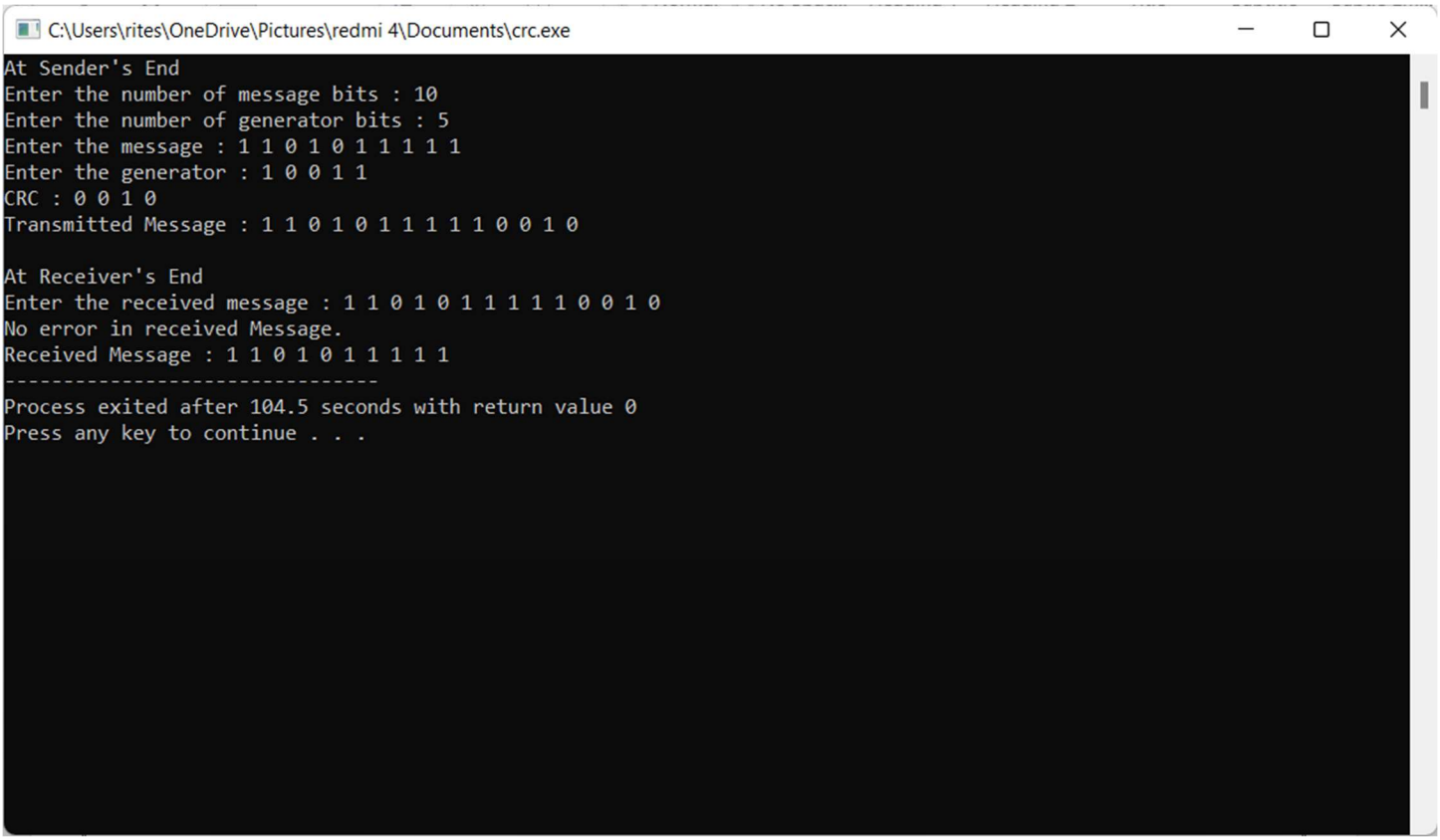


PROGRAM 2

2. Simulate and implement stop and wait protocol for noisy channel.

```
#include<bits/stdc++.h>
#include <unistd.h>
using namespace std;
int main()
{
   int frames;
   cout<<"input number of frames-->";
   cin>>frames;
   int i=1;
   int sleepTime=2;
   srand(time(0));
   while(frames>0)
   {
      cout<<"sending frame is -- "<<i<<"\n\n";
      int x=rand()%10;
      if(x==5 || x==0 || x==9)
      {
         cout<<"\tERROR ERROR in sending \n";
         cout<<"\twaiting for "<<sleepTime-1<<" seconds\n\n";
         usleep(2000000);
         cout<<"sending frame -- "<<i<<" again "<<endl;
         // int x=rand()%10;
      }
      cout<<"acknowledgement for frame -- "<<i<<"\n\n";
```

```
      frames--;
      i++;
   }

   cout<<"\tEnd of STOP n WAIT protocol \n";
}
```



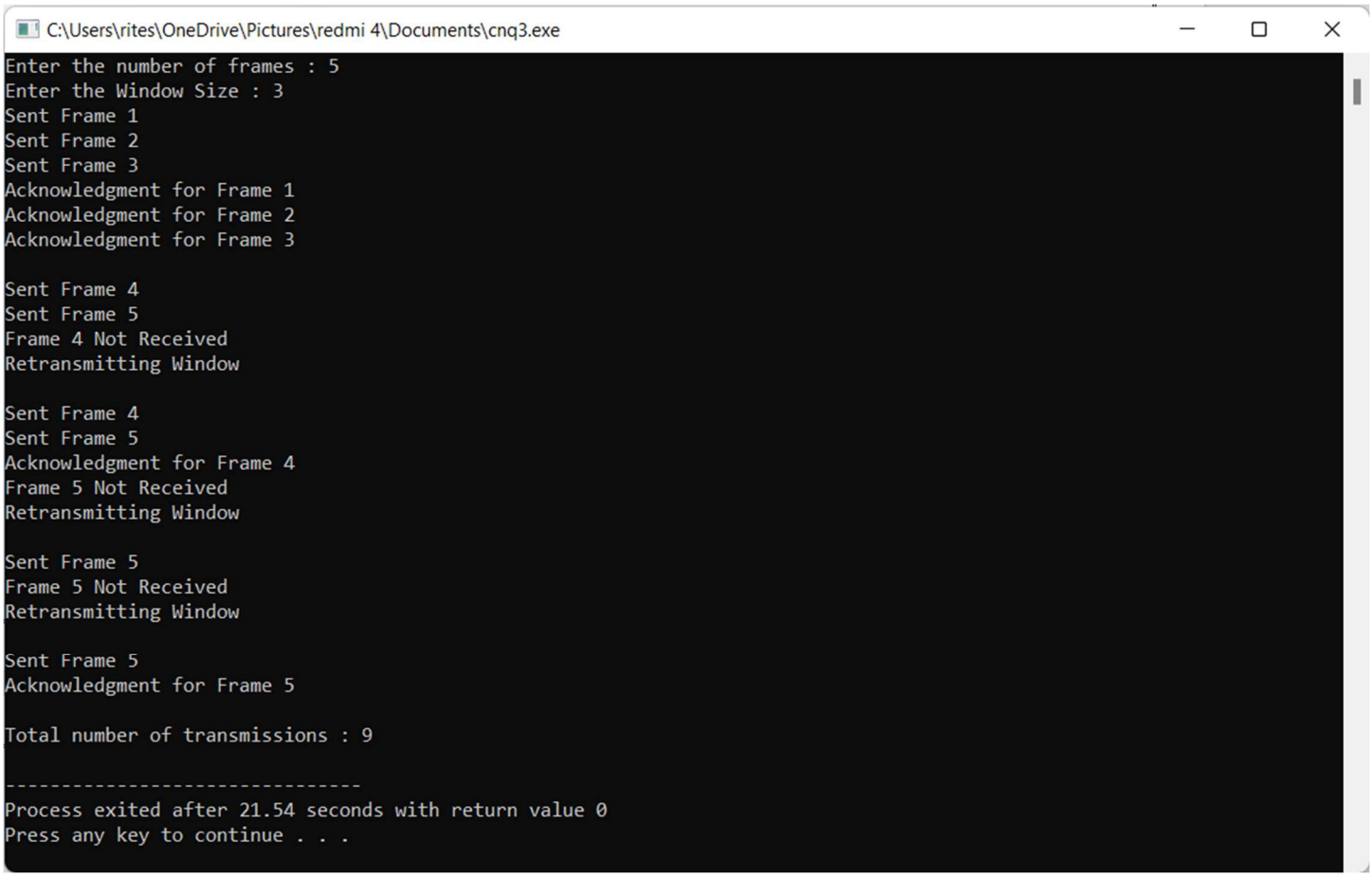
Program 3

Simulate and implement go back n sliding window protocol.

```
#include<iostream>
#include<ctime>
#include<cstdlib>
using namespace std;
int main()
{
 int nf,N;
 int no_tr=0;
 srand(time(NULL));
 cout<<"Enter the number of frames : ";
 cin>>nf;
 cout<<"Enter the Window Size : ";
 cin>>N;
 int i=1;
 while(i<=nf)
 {
    int x=0;
    for(int j=i;j<i+N && j<=nf;j++)
    {
       cout<<"Sent Frame "<<j<<endl;
       no_tr++;
    }
    for(int j=i;j<i+N && j<=nf;j++)
    {
       int flag = rand()%2;
```

```
        if(!flag)
          {
            cout<<"Acknowledgment for Frame "<<j<<endl;
            x++;
          }
        else
          {   cout<<"Frame "<<j<<" Not Received"<<endl;
            cout<<"Retransmitting Window"<<endl;
            break;
          }
    }
    cout<<endl;
    i+=x;
 }
 cout<<"Total number of transmissions : "<<no_tr<<endl;
 return 0;
}
```

OUTPUT:


C:\Users\rites\OneDrive\Pictures\redmi 4\Documents\cnq3.exe
Enter the number of frames : 5
Enter the Window Size : 3
Sent Frame 1
Sent Frame 2
Sent Frame 3
Acknowledgment for Frame 1
Acknowledgment for Frame 2
Acknowledgment for Frame 3
Sent Frame 4
Sent Frame 5
Frame 4 Not Received
Retransmitting Window
Sent Frame 4
Sent Frame 5
Acknowledgment for Frame 4
Frame 5 Not Received
Retransmitting Window
Sent Frame 5
Frame 5 Not Received
Retransmitting Window
Sent Frame 5
Acknowledgment for Frame 5
Total number of transmissions : 9
Process exited after 21.54 seconds with return value 0
Press any key to continue . . .

Program 4:

Simulate and implement selective repeat sliding window protocol.

```
# include <iostream>
# include <stdlib.h>
# include <time.h>
# include <math.h>
using namespace std;
class sel_repeat
{
 private:
  int fr_send_at_instance;
  int arr[TOT_FRAMES];
  int send[FRAMES_SEND];
  int rcvd[FRAMES_SEND];
  char rcvd_ack[FRAMES_SEND];
  int sw;
  int rw; // tells expected frame
 public:
  void input();
  void sender(int);
  void reciever(int);
};
void sel_repeat :: input()
{
 int n;  // no of bits for the frame
 int m;  // no of frames from n bits
 cout << "Enter the no of bits for the sequence number ";
```

```
 cin >> n;
 m = pow (2 , n);
 int t = 0;
 fr_send_at_instance = (m / 2);
 for (int i = 0 ; i < TOT_FRAMES ; i++)
 {
  arr[i] = t;
  t = (t + 1) % m;
 }
 for (int i = 0 ; i < fr_send_at_instance ; i++)
 {
  send[i] = arr[i];
  rcvd[i] = arr[i];
  rcvd_ack[i] = 'n';
 }
 rw = sw = fr_send_at_instance;
 sender(m);
}
void sel_repeat :: sender(int m)
{
 for (int i = 0 ; i < fr_send_at_instance ; i++)
 {
  if ( rcvd_ack[i] == 'n' )
    cout << " SENDER   : Frame " << send[i] << " is sent\n";
 }
 reciever (m);
}
void sel_repeat :: reciever(int m)
{
```

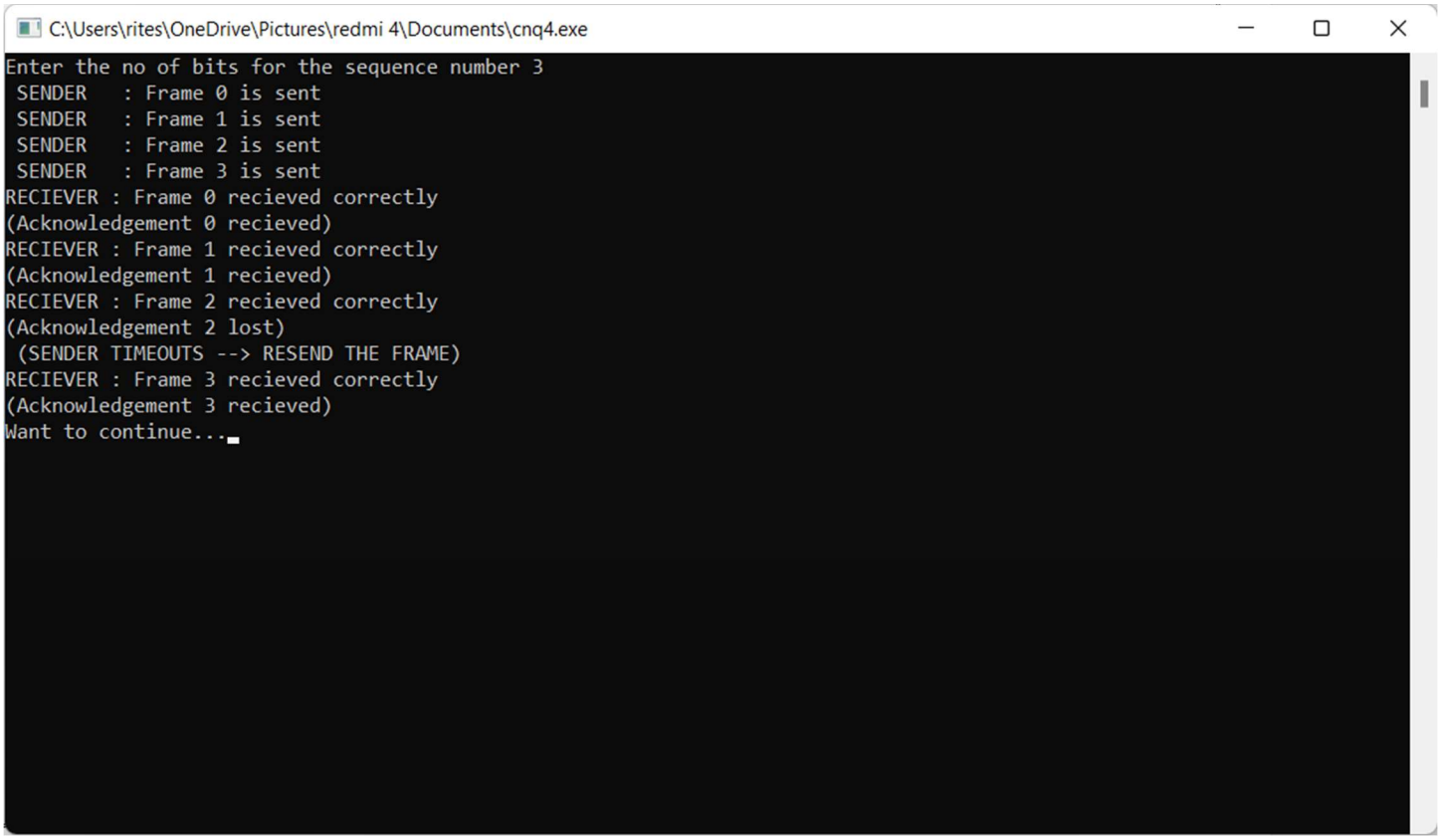
```
time_t t;
int f;
int f1;
int a1;
char ch;
int j=0;
int i=0;
srand((unsigned) time(&t));
for (i = 0 ; i < fr_send_at_instance ; i++)
{
 if (rcvd_ack[i] == 'n')
 {
  f = rand() % 10;
  // if = 5 frame is discarded for some reason
  // else frame is correctly recieved
  if (f != 5)
  {
   for (j = 0 ; j < fr_send_at_instance ; j++)
    if (rcvd[j] == send[i])
    {
     cout << "RECIEVER : Frame " << rcvd[j] << " recieved correctly\n";
     rcvd[j] = arr[rw];
     rw = (rw + 1) % m;
     break;
    }
   if (j == fr_send_at_instance)
    cout << "RECIEVER : Duplicate frame " << send[i] << " discarded\n";
   a1 = rand() % 5;
    // if a1 == 3 then ack is lost
```

```
 //          else recieved
if (a1 == 3)
{
 cout << "(Acknowledgement " << send[i] << " lost)\n";
 cout << " (SENDER TIMEOUTS --> RESEND THE FRAME)\n";
 rcvd_ack[i] = 'n';
}
else
{
 cout << "(Acknowledgement " << send[i] << " recieved)\n";
 rcvd_ack[i] = 'p';
}
}
else
{
 int ld = rand() % 2;

 // if = 0 then frame damaged
 // else frame lost
 if (ld == 0)
 {
  cout << "RECIEVER : Frame " << send[i] << " is damaged\n";
  cout << "RECIEVER : Negative acknowledgement " << send[i]  << " sent\n";
 }
 else
 {
  cout << "RECIEVER : Frame " << send[i] << " is lost\n";
  cout << " (SENDER TIMEOUTS --> RESEND THE FRAME)\n";
 }
```

```
   rcvd_ack[i] = 'n';
  }
 }
}
for ( int j = 0 ; j < fr_send_at_instance ; j++)
{
 if (rcvd_ack[j] == 'n')
  break;
}
i = 0 ;
for (int k = j ; k < fr_send_at_instance ; k++)
{
 send[i] = send[k];

 if (rcvd_ack[k] == 'n')
  rcvd_ack[i] = 'n';
 else
  rcvd_ack[i] = 'p';
 i++;
}
if ( i != fr_send_at_instance )
{
 for ( int k = i ; k < fr_send_at_instance ; k++)
 {
  send[k] = arr[sw];
  sw = (sw + 1) % m;
  rcvd_ack[k] = 'n';
 }}
cout << "Want to continue...";
```

```
 cin >> ch;
 cout << "\n";
 if (ch == 'y')
  sender(m);
 else
  exit(0);
}
int main()
{
 sel_repeat sr;
 sr.input();
 return 0;
}
```

OUTPUT :



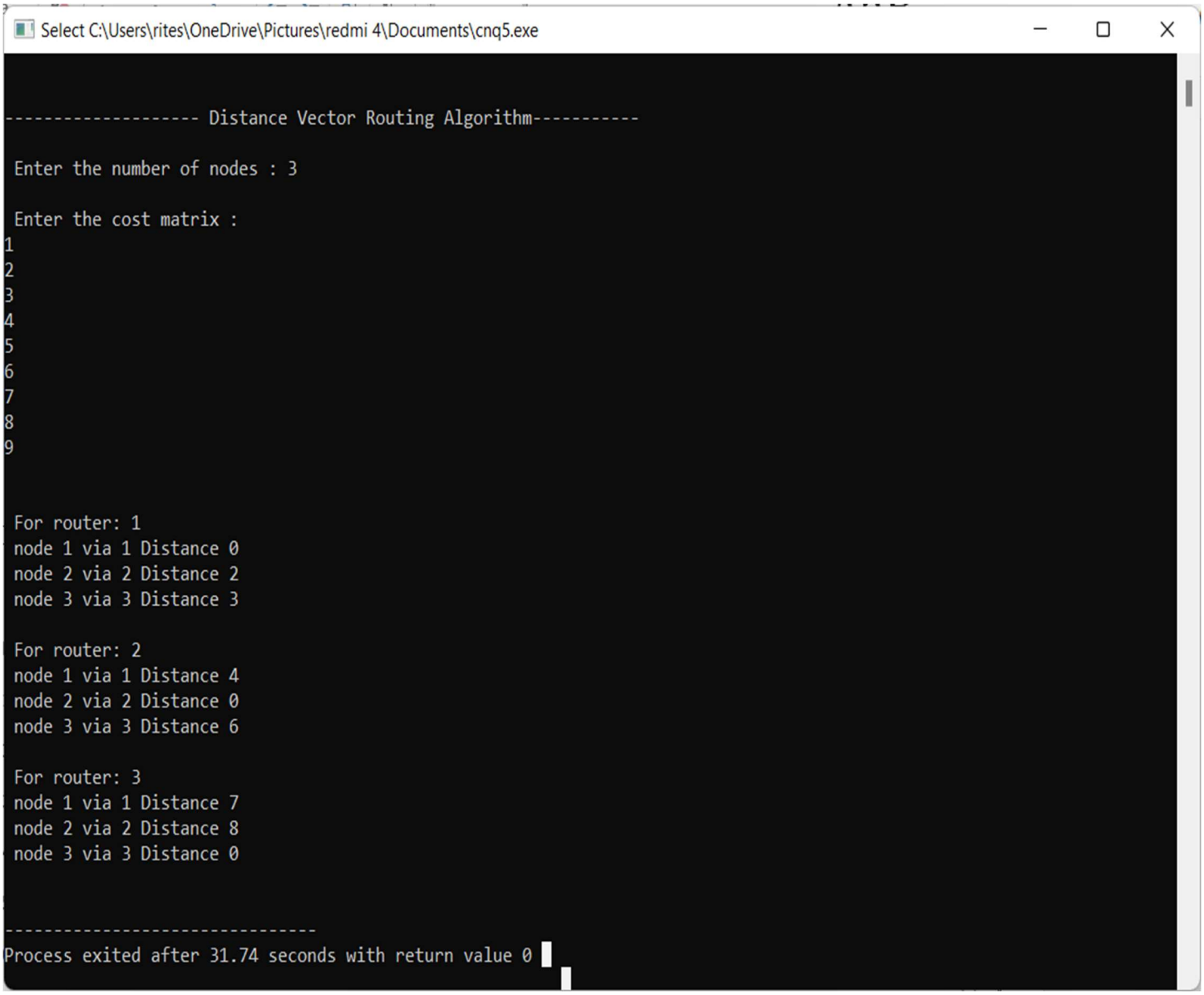
Program 5

Simulate and implement distance vector routing algorithm.

```
#include<stdio.h>
#include<iostream>
using namespace std;
struct node
{
   unsigned dist[6];
   unsigned from[6];
}DVR[10];
int main()
{
   cout<<"\n\n---------- Distance Vector Routing Algorithm---------- ";
   int costmat[6][6];
   int nodes, i, j, k;
   cout<<"\n\n Enter the number of nodes : ";
   cin>>nodes; //Enter the nodes
   cout<<"\n Enter the cost matrix : \n" ;
   for(i = 0; i < nodes; i++)
    {
      for(j = 0; j < nodes; j++)
      {
         cin>>costmat[i][j];
         costmat[i][i] = 0;
         DVR[i].dist[j] = costmat[i][j]; //initialise the distance equal to cost matrix
         DVR[i].from[j] = j;
      }
```

```
  }
        for(i = 0; i < nodes; i++) //We choose arbitary vertex k and we calculate the
        //direct distance from the node i to k using the cost matrix and add the distance from k to node j
        for(j = i+1; j < nodes; j++)
        for(k = 0; k < nodes; k++)
           if(DVR[i].dist[j] > costmat[i][k] + DVR[k].dist[j])
           {  //We calculate the minimum distance
              DVR[i].dist[j] = DVR[i].dist[k] + DVR[k].dist[j];
              DVR[j].dist[i] = DVR[i].dist[j];
              DVR[i].from[j] = k;
              DVR[j].from[i] = k;
           }
     for(i = 0; i < nodes; i++)
     {
        cout<<"\n\n For 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<<" \n\n ";
  return 0;
}
```

OUTPUT:


Select C:\Users\rites\OneDrive\Pictures\redmi 4\Documents\cnq5.exe
---------- Distance Vector Routing Algorithm----------
Enter the number of nodes : 3
Enter the cost matrix :
1
2
3
4
5
6
7
8
9
For router: 1
node 1 via 1 Distance 0
node 2 via 2 Distance 2
node 3 via 3 Distance 3
For router: 2
node 1 via 1 Distance 4
node 2 via 2 Distance 0
node 3 via 3 Distance 6
For router: 3
node 1 via 1 Distance 7
node 2 via 2 Distance 8
node 3 via 3 Distance 0
Process exited after 31.74 seconds with return value 0

Program 6

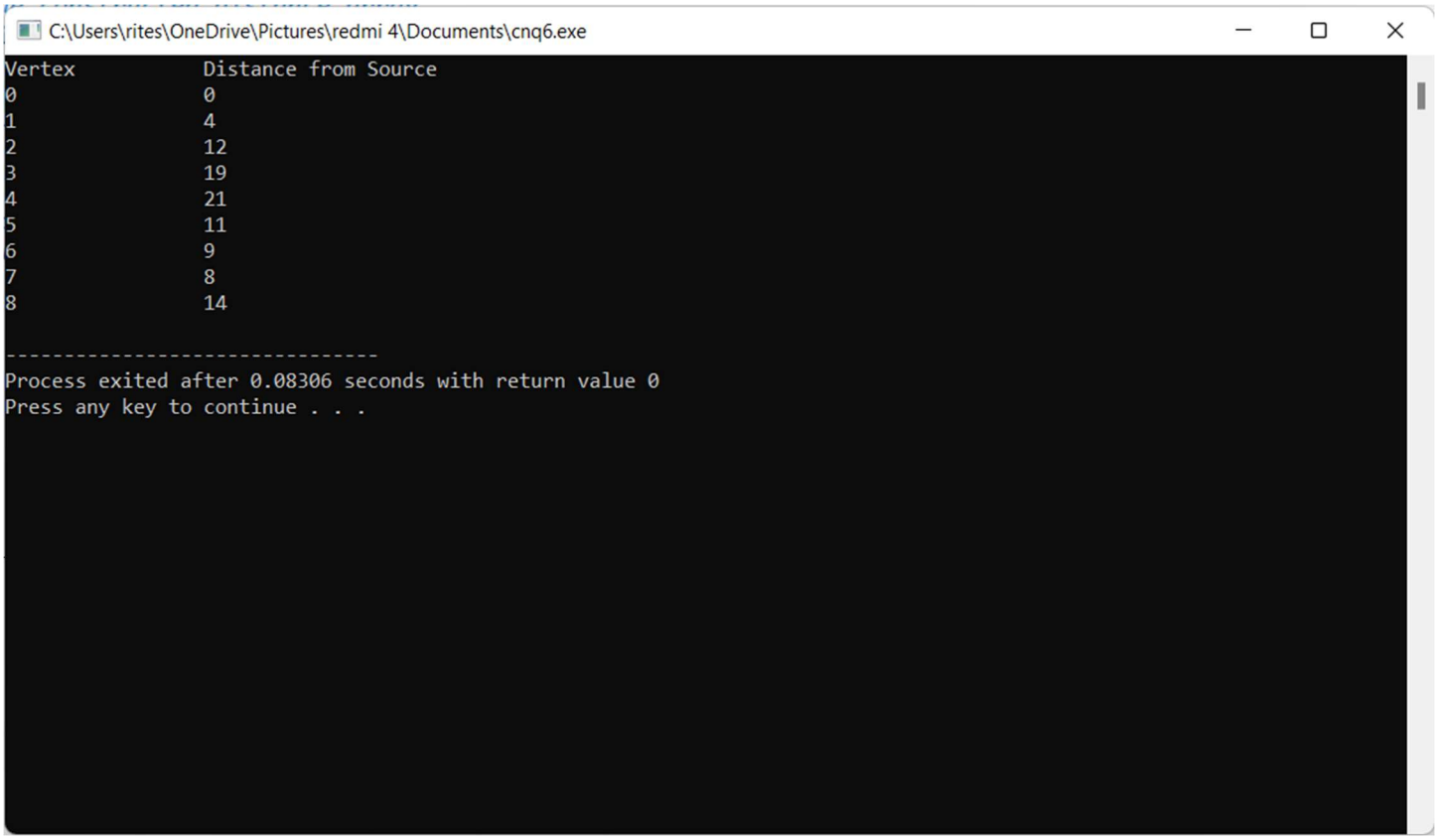
Simulate and implement Dijkstra algorithm for shortest path routing.

```
#include <limits.h>

#include <stdio.h>


// Number of vertices in the graph

#define V 9


// A utility function to find the vertex with minimum distance value, from

// the set of vertices not yet included in shortest path tree

int minDistance(int dist[], bool sptSet[])

{

   // Initialize min value

   int min = INT_MAX, min_index;


   for (int v = 0; v < V; v++)

      if (sptSet[v] == false && dist[v] <= min)

         min = dist[v], min_index = v;


   return min_index;

}


// A utility function to print the constructed distance array

void printSolution(int dist[])

{

   printf("Vertex \t\t Distance from Source\n");

   for (int i = 0; i < V; i++)

      printf("%d \t\t %d\n", i, dist[i]);
```

```
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
   int dist[V]; // The output array.  dist[i] will hold the shortest
   // distance from src to
   // Initialize all distances as INFINITE and stpSet[] as false I

   bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;

   // Distance of source vertex from itself is always 0
   dist[src] = 0;

   // Find shortest path for all vertices
   for (int count = 0; count < V - 1; count++) {
      // Pick the minimum distance vertex from the set of vertices not
      // yet processed. u is always equal to src in the first iteration.
      int u = minDistance(dist, sptSet);

      // Mark the picked vertex as processed
      sptSet[u] = true;

      // Update dist value of the adjacent vertices of the picked vertex.
      for (int v = 0; v < V; v++)
```

```
            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
        }

        // print the constructed distance array
        printSolution(dist);
}

// driver program to test above function
int main()
{
        /* Let us create the example graph discussed above */
        int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                            { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                            { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                            { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                            { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                            { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                            { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                            { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                            { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
        dijkstra(graph, 0);

        return 0;
}
```

OUTPUT:


C:\Users\rites\OneDrive\Pictures\redmi 4\Documents\cnq6.exe
Vertex Distance from Source
0 0
1 4
2 12
3 19
4 21
5 11
6 9
7 8
8 14

Process exited after 0.08306 seconds with return value 0
Press any key to continue . . .