SOURCE CODE

SENDER (sender.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <time.h>


//#define number_of_packets 6
//#define maximum_sequence_number 2 // must be greater than 1/2 the window size
//#define window_size 3

int packetsize; //packet size
int timeoutinterval; // time out interval in seconds

void printsw(int sendingwindow[], int window_size)
{

int i;

printf("Current window = [");
for(i=0;i<window_size;i++)
{
  if(i==window_size-1)
  {
    printf("%d",sendingwindow[i]);
  }
  else
  {
    printf("%d,",sendingwindow[i]);
  }
}
printf("]\n");

}


void slidewindow(int sendingwindow[], int window_size, int np)
{

int i;

for(i=0;i<window_size-1;i++)
{
  sendingwindow[i] = sendingwindow[i+1];
}
sendingwindow[window_size-1] = np; // put new packet into rightmost frame of the sliding window

printsw(sendingwindow, window_size);
```

```
}


void invertbits(char *a, char *b)
{
  int len = strlen(a);
  int i;
  for(i=0;i<len;i++)
  {
    if(a[i] == '0') b[i] = 'f';
    else if(a[i] == '1') b[i] = 'e';
    else if(a[i] == '2') b[i] = 'd';
    else if(a[i] == '3') b[i] = 'c';
    else if(a[i] == '4') b[i] = 'b';
    else if(a[i] == '5') b[i] = 'a';
    else if(a[i] == '6') b[i] = '9';
    else if(a[i] == '7') b[i] = '8';
    else if(a[i] == '8') b[i] = '7';
    else if(a[i] == '9') b[i] = '6';
    else if(a[i] == 'a') b[i] = '5';
    else if(a[i] == 'b') b[i] = '4';
    else if(a[i] == 'c') b[i] = '3';
    else if(a[i] == 'd') b[i] = '2';
    else if(a[i] == 'e') b[i] = '1';
    else b[i] = '0';
  }
  b[len] = '\0';
}


char* pack(char *a, char *c, char *sumbuf, int errorflag)
{
  //printf("---PACKING---\n");
  //printf("%s\n",a);
  int len = strlen(a);
  int i;
  int val;
  char buf[3]; // val can be max 255 and 255 = ff in hex so only 2 characters needed to store
  int sum = 0;
  int sumlength;
  int msgsize = 3 * len; // size of buf times number of characters
  char *b;
  b = malloc(msgsize);
  b[0] = '\0';
  for(i=0;i<len;i++)
  {
    val = a[i]; //  ASCII value of character. Eg. 'h' has value 104
    sum = sum + val;
    sprintf(buf,"%x",val);
    strcat(b,buf);
  }
  sprintf(sumbuf,"%x",sum);
  //printf("%s\n",sumbuf);
  sumlength = strlen(sumbuf);
  //c = malloc(sumlength);
  if(errorflag == 0) invertbits(sumbuf,c);
  else c = sumbuf;
  //printf("%s\n",c);
  b = realloc(b,msgsize + sumlength + 1);
  strcat(b,"|");
  strcat(b,c);
  //printf("Transmitted Packet:");
  //printf("%s\n",b);
  return b;
}

void perfectimp(int numpackets, int msn, int window_size, int sock, int packetarray[], char
abc[][packetsize+1], int sw[])
{
  printsw(sw, window_size);

  char *sumbuf = (char*) malloc(100);
  char *checksum = (char*) malloc(100);
  char *transmitmsg;
  int sumlen; // length of sum and checksum for that particular packet
  int tmlen; //length of the transmitted msg
  int msglen; //length of string msg eg.shya = 4

  int numacks = 0; //the total number of acknowledgements received
  int packid; //the next packet to send
  int ackbuf; //variable to hold the ack number;
  int nextpacketindex = window_size; //variable to track the index of the next packet to enter the
sender window
  int nextpacket; //the next packet to enter the sender window
  int slidecount = numpackets - window_size;
  int nextpackidindex = 0; // variable to keep track of the index of the next packet to send in the
sliding window
  int available_frames = window_size;
  int numpacketssent = 0; // variable to keep track of the number of packets sent
  int packetflag = 0; // variable to control the transmission of packet information like number of
packets and max seq number
  int j = 0;
  int acklostindex = -1;

  nextpacket = packetarray[nextpacketindex];

  while(numacks < numpackets)
  {
    if(packetflag == 0)
    {
      write(sock,&numpackets,sizeof(numpackets));
      write(sock,&msn,sizeof(msn));
      write(sock,&acklostindex,sizeof(acklostindex));
      packetflag = 1;
    }

    while(available_frames > 0 && numpacketssent < numpackets) // send all packets in the sending
window
    {
      packid = sw[nextpackidindex];
      printf("Packet %d sent\n",packid);
      msglen = strlen(abc[j]);
      transmitmsg = pack(abc[j],checksum,sumbuf,0);
      //printf("%s\n",transmitmsg);
      tmlen = strlen(transmitmsg);
      //printf("%d\n",tmlen);
      transmitmsg[tmlen] = '\0';
      sumlen = strlen(sumbuf);
      write(sock,&packid,sizeof(packid));
      write(sock,&tmlen,sizeof(tmlen)); //send total length of transmitting msg
      write(sock,transmitmsg,tmlen+1); // send the msg
      write(sock,&sumlen,sizeof(sumlen)); //send size of sum/checksum
      write(sock,sumbuf,sumlen+1); //send the sum
      write(sock,checksum,sumlen+1); // send the checksum
      write(sock,&msglen,sizeof(msglen)); // length of msg eg.shya = 4
      j++;
      available_frames--;
      numpacketssent++;
      if(nextpackidindex < window_size-1)
      {
        nextpackidindex++;
      }
    }
    read(sock,&ackbuf,sizeof(ackbuf));
    printf("Ack %d received\n",ackbuf);
    numacks++;
    available_frames++;
    if(slidecount > 0)
    {
      slidewindow(sw, window_size, nextpacket);
      slidecount--;
    }
    else
    {
    if(numacks < numpackets)
    {
      printsw(sw, window_size);
    }
  }

    if(nextpacketindex+1 < numpackets)
    {
      nextpacketindex++;
      nextpacket = packetarray[nextpacketindex];
    }
  }

printf("\nNumber of packets sent:%d\n",numpacketssent);

free(transmitmsg);
free(sumbuf);
free(checksum);

}
```

```
void damagedpacket(int numpackets, int msn, int window_size, int sock, int packetarray[], char
abc[][packetsize+1], int sw[], int dmgpack)
{
  printsw(sw, window_size);

  char *sumbuf = (char*) malloc(100);
  char *checksum = (char*) malloc(100);
  char *transmitmsg;
  int sumlen; // length of sum and checksum for that particular packet
  int tmlen; //length of the transmitted msg
  int msglen; //length of string msg eg.shya = 4

  int numacks = 0; //the total number of acknowledgements received
  int packid; //the next packet to send
  int ackbuf; //variable to hold the ack number;
  int nextpacketindex = window_size; //variable to track the index of the next packet to enter the
sender window
  int nextpacket; //the next packet to enter the sender window
  int slidecount = numpackets - window_size;
  int nextpackidindex = 0; // variable to keep track of the index of the next packet to send in the
sliding window
  int available_frames = window_size;
  int numpacketssent = 0; // variable to keep track of the number of packets sent
  int packetflag = 0; // variable to control the transmission of packet information like number of
packets and max seq number
  int j = 0;
  int resendflag = 0;
  int errorpackindex; // to keep track of j value with error
  int resendcount; // to count number of packets to resend
  int errorpackid; //to keep track of the erroneous packet
  int npidtoretransmit = 0;
  int numpackstoresend = 0;
  int acklostindex = -1;

  nextpacket = packetarray[nextpacketindex];

  while(numacks < numpackets)
  {
    if(packetflag == 0)
    {
      write(sock,&numpackets,sizeof(numpackets));
      write(sock,&msn,sizeof(msn));
      write(sock,&acklostindex,sizeof(acklostindex));
      packetflag = 1;
    }

    while(available_frames > 0 && numpacketssent < numpackets) // send all packets in the sending
window
    {
      packid = sw[nextpackidindex];
      if(resendflag == 1)
      {
        sleep(timeoutinterval);
        printf("Packet %d timed out\n", packid);
        printf("Packet %d re-transmitted\n",packid);
      }
      else printf("Packet %d sent\n",packid);
      msglen = strlen(abc[j]);
      if(resendflag == 1)
      {
       transmitmsg = pack(abc[j],checksum,sumbuf,0);
       numpackstoresend--;

      }
      else
      {
        if(numpacketssent == dmgpack)
        {
          transmitmsg = pack(abc[j],checksum,sumbuf,1);
          errorpackindex = j;
          resendcount = numpacketssent;
          //printf("Resendcount:%d\n",resendcount);
          errorpackid = packid;
          //printf("EPID:%d\n",errorpackid);
          npidtoretransmit = nextpackidindex;
          //printf("NPIDIN:%d\n",npidtoretransmit);
        }
        else transmitmsg = pack(abc[j],checksum,sumbuf,0);
      }
      //printf("%s\n",transmitmsg);
      tmlen = strlen(transmitmsg);
      //printf("%d\n",tmlen);
      transmitmsg[tmlen] = '\0';
      sumlen = strlen(sumbuf);
      write(sock,&packid,sizeof(packid));
      write(sock,&tmlen,sizeof(tmlen)); //send total length of transmitting msg
      write(sock,transmitmsg,tmlen+1); // send the msg
      write(sock,&sumlen,sizeof(sumlen)); //send size of sum/checksum
      write(sock,sumbuf,sumlen+1); //send the sum
      write(sock,checksum,sumlen+1); // send the checksum
      write(sock,&msglen,sizeof(msglen)); // length of msg eg.shya = 4
      j++;
      available_frames--;
      numpacketssent++;
      //printf("NPS:%d\n",numpacketssent);
      if(nextpackidindex < window_size-1)
      {
        nextpackidindex++;
      }
    }
    if(numacks != dmgpack || resendflag == 1)
    {
      if(numpackstoresend == 0) resendflag = 0;
      read(sock,&ackbuf,sizeof(ackbuf));
      printf("Ack %d received\n",ackbuf);
      numacks++;
      available_frames++;
      if(slidecount > 0)
      {
          slidewindow(sw, window_size, nextpacket);
          slidecount--;
          npidtoretransmit--;
        }
        else
        {
          if(numacks < numpackets)
          {
            printsw(sw, window_size);
          }
        }
        if(nextpacketindex+1 < numpackets)
        {
          nextpacketindex++;
          nextpacket = packetarray[nextpacketindex];
          //printf("NEXTPACK:%d\n",nextpacket);
        }
      }

      else
      {
       resendflag = 1;
       available_frames = numpacketssent - numacks;
       numpackstoresend = numpacketssent - resendcount;
       //printf("NPTRS:%d\n",numpackstoresend);
       numpacketssent = resendcount;
       j = errorpackindex;
       nextpackidindex = npidtoretransmit;
      }

  }

printf("\nNumber of packets sent:%d\n",numpacketssent);

free(transmitmsg);
free(sumbuf);
free(checksum);

}

void lostpacket(int numpackets, int msn, int window_size, int sock, int packetarray[], char
abc[][packetsize+1], int sw[], int lostpack)
{
  printsw(sw, window_size);

  char *sumbuf = (char*) malloc(100);
  char *checksum = (char*) malloc(100);
  char *transmitmsg;
  int sumlen; // length of sum and checksum for that particular packet
  int tmlen; //length of the transmitted msg
  int msglen; //length of string msg eg.shya = 4

  int numacks = 0; //the total number of acknowledgements received
  int packid; //the next packet to send
  int ackbuf; //variable to hold the ack number;
  int nextpacketindex = window_size; //variable to track the index of the next packet to enter the
sender window
  int nextpacket; //the next packet to enter the sender window
  int slidecount = numpackets - window_size;
  int nextpackidindex = 0; // variable to keep track of the index of the next packet to send in the
sliding window
  int available_frames = window_size;
  int numpacketssent = 0; // variable to keep track of the number of packets sent
  int packetflag = 0; // variable to control the transmission of packet information like number of
packets and max seq number
  int j = 0;
  int resendflag = 0;
  int errorpackindex; // to keep track of j value with error
  int resendcount; // to count number of packets to resend
  int errorpackid; //to keep track of the erroneous packet
  int npidtoretransmit = 0;
  int numpackstoresend = 0;
  int acklostindex = -1;

  nextpacket = packetarray[nextpacketindex];

  while(numacks < numpackets)
  {
    if(packetflag == 0)
    {
      write(sock,&numpackets,sizeof(numpackets));
      write(sock,&msn,sizeof(msn));
      write(sock,&acklostindex,sizeof(acklostindex));
      packetflag = 1;
    }

    while(available_frames > 0 && numpacketssent < numpackets) // send all packets in the sending
window
    {
      packid = sw[nextpackidindex];

      if(resendflag == 1)
      {
        sleep(timeoutinterval);
        printf("Packet %d timed out\n", packid);
        printf("Packet %d re-transmitted\n",packid);
      }
      else printf("Packet %d sent\n",packid);

      msglen = strlen(abc[j]);

      if(resendflag == 1)
      {
       transmitmsg = pack(abc[j],checksum,sumbuf,0);
       numpackstoresend--;

      }
      else
      {
        if(numpacketssent == lostpack)
        {
          //transmitmsg = pack(abc[j],checksum,sumbuf,1);
          errorpackindex = j;
          resendcount = numpacketssent;
          //printf("Resendcount:%d\n",resendcount);
          errorpackid = packid;
          //printf("EPID:%d\n",errorpackid);
          npidtoretransmit = nextpackidindex;
          //printf("NPIDIN:%d\n",npidtoretransmit);
        }
        else transmitmsg = pack(abc[j],checksum,sumbuf,0);
      }

      if(numpacketssent != lostpack || resendflag == 1)
      {
      tmlen = strlen(transmitmsg);
      transmitmsg[tmlen] = '\0';
      sumlen = strlen(sumbuf);
      write(sock,&packid,sizeof(packid));
      write(sock,&tmlen,sizeof(tmlen)); //send total length of transmitting msg
      write(sock,transmitmsg,tmlen+1); // send the msg
      write(sock,&sumlen,sizeof(sumlen)); //send size of sum/checksum
      write(sock,sumbuf,sumlen+1); //send the sum
      write(sock,checksum,sumlen+1); // send the checksum
      write(sock,&msglen,sizeof(msglen)); // length of msg eg.shya = 4
      }
      j++;
      available_frames--;
      numpacketssent++;
      //printf("NPS:%d\n",numpacketssent);
      if(nextpackidindex < window_size-1)
      {
        nextpackidindex++;
      }
    }
    if(numacks != lostpack || resendflag == 1)
    {
      if(numpackstoresend == 0) resendflag = 0;
      read(sock,&ackbuf,sizeof(ackbuf));
      printf("Ack %d received\n",ackbuf);
      numacks++;
      available_frames++;
      if(slidecount > 0)
      {
        slidewindow(sw, window_size, nextpacket);
        slidecount--;
        npidtoretransmit--;
      }
      else
      {
        if(numacks < numpackets)
        {
          printsw(sw, window_size);
        }
      }
```

```
        if(nextpacketindex+1 < numpackets)
        {
          nextpacketindex++;
          nextpacket = packetarray[nextpacketindex];
          //printf("NEXTPACK:%d\n",nextpacket);
        }
      }

      else
      {
        resendflag = 1;
        available_frames = numpacketssent - numacks;
        numpackstoresend = numpacketssent - resendcount;
        //printf("NPTRS:%d\n",numpackstoresend);
        numpacketssent = resendcount;
        j = errorpackindex;
        nextpackidindex = npidtoretransmit;
      }

    }

printf("\nNumber of packets sent:%d\n",numpacketssent);

free(transmitmsg);
free(sumbuf);
free(checksum);

}

void lostacknowledgement(int numpackets, int msn, int window_size, int sock, int packetarray[], char
abc[][packetsize+1], int sw[], int lostack)
{
  printsw(sw, window_size);

  char *sumbuf = (char*) malloc(100);
  char *checksum = (char*) malloc(100);
  char *transmitmsg;
  int sumlen; // length of sum and checksum for that particular packet
  int tmlen; //length of the transmitted msg
  int msglen; //length of string msg eg.shya = 4

  int numacks = 0; //the total number of acknowledgements received
  int packid; //the next packet to send
  int ackbuf; //variable to hold the ack number;
  int nextpacketindex = window_size; //variable to track the index of the next packet to enter the
sender window
  int nextpacket; //the next packet to enter the sender window
  int slidecount = numpackets - window_size;
  int nextpackidindex = 0; // variable to keep track of the index of the next packet to send in the
sliding window
  int available_frames = window_size;
  int numpacketssent = 0; // variable to keep track of the number of packets sent
  int packetflag = 0; // variable to control the transmission of packet information like number of
packets and max seq number
  int j = 0;
  int acklostindex = lostack;
  int resendflag = 0;
  int errorpackindex;
  int errorpackid;

  nextpacket = packetarray[nextpacketindex];

  while(numacks < numpackets)
  {
    if(packetflag == 0)
    {
      write(sock,&numpackets,sizeof(numpackets));
      write(sock,&msn,sizeof(msn));
      write(sock,&acklostindex,sizeof(acklostindex));
      packetflag = 1;
    }

    while(available_frames > 0 && numpacketssent < numpackets) // send all packets in the sending
window
    {
      if(resendflag == 1)
      {
        packid = errorpackid;
      }

      else
      {
        packid = sw[nextpackidindex];
        printf("Packet %d sent\n",packid);
      }

      if(resendflag == 1)
      {
        j = errorpackindex;
        msglen = strlen(abc[j]);
        transmitmsg = pack(abc[j],checksum,sumbuf,0);
        resendflag = 0;
      }
      else
      {
        msglen = strlen(abc[j]);
        transmitmsg = pack(abc[j],checksum,sumbuf,0);
      }

      if(numpacketssent == lostack)
      {
       errorpackindex = j;
       errorpackid = packid;
      }
      //printf("%s\n",transmitmsg);
      tmlen = strlen(transmitmsg);
      //printf("%d\n",tmlen);
      transmitmsg[tmlen] = '\0';
      sumlen = strlen(sumbuf);
      write(sock,&packid,sizeof(packid));
```

```
        write(sock,&tmlen,sizeof(tmlen)); //send total length of transmitting msg
        write(sock,transmitmsg,tmlen+1); // send the msg
        write(sock,&sumlen,sizeof(sumlen)); //send size of sum/checksum
        write(sock,sumbuf,sumlen+1); //send the sum
        write(sock,checksum,sumlen+1); // send the checksum
        write(sock,&msglen,sizeof(msglen)); // length of msg eg.shya = 4
        j++;
        available_frames--;
        numpacketssent++;
        if(nextpackidindex < window_size-1)
        {
          nextpackidindex++;
        }
      }

      if(numacks == lostack)
      {
        //printf("LOSTACK:%d\n",lostack);
        sleep(timeoutinterval);
        printf("Packet %d timed out\n", lostack);
        printf("Packet %d re-transmitted\n",lostack);
        resendflag = 1;
        available_frames++;
        numpacketssent--;
        j--;
      }

      read(sock,&ackbuf,sizeof(ackbuf));
      printf("Ack %d received\n",ackbuf);
      numacks++;
      available_frames++;
      if(slidecount > 0)
      {
        slidewindow(sw, window_size, nextpacket);
        slidecount--;
      }
      else
      {
      if(numacks < numpackets)
      {
        printsw(sw, window_size);
      }
    }

      if(nextpacketindex+1 < numpackets)
      {
        nextpacketindex++;
        nextpacket = packetarray[nextpacketindex];
      }
    }

  printf("\nNumber of packets sent:%d\n",numpacketssent);

  free(transmitmsg);
  free(sumbuf);
```

```
free(checksum);

}

int main(int argc, char *argv[])
{

if(argc != 2)
{
  printf("More arguments expected\n");
  exit(0);
}

int sock = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in s_address;
s_address.sin_family = AF_INET;
s_address.sin_port = htons(51374);
s_address.sin_addr.s_addr = INADDR_ANY;
if (connect(sock, (struct sockaddr *) &s_address, sizeof(s_address)) < 0)
{
   printf("Cannot connect\n");
   exit(0);
}

/*** File operations section ***/
FILE *fp;
char ch;
int size;
char *buf;
//packetsize = 128; // size of a packet in bytes
int nop; // no of packets needed
int window_size;
int maximum_sequence_number;
int choice;
int errchoice;
int uichoice;

struct timeval start,end;

while(choice!=5)
{
  printf("\nUSER INPUT\n");
  printf("----------\n");
  printf("1.Packet size\n");
  printf("2.Timeout interval (in seconds)\n");
  printf("3.Size of sliding window\n");
  printf("4.Max sequnece number (Min is 0)\n");
  printf("5.Exit\n");
  printf("Enter your choice:");
  scanf("%d",&choice);
  switch(choice)
  {
    case 1:printf("Enter the size of each packet:");
         scanf("%d",&packetsize);
```

```
            break;

      case 2:printf("Enter the timeout interval:");
            scanf("%d",&timeoutinterval);
            break;

      case 3:printf("Enter the size of the sliding window:");
            scanf("%d",&window_size);
            break;

      case 4:printf("Enter the maximum sequence number:");
            scanf("%d",&maximum_sequence_number);
            break;

      case 5:break;

      default:printf("Invalid choice\n");
             break;
   }
 }

 char fnamebuf[strlen(argv[1])];
 strcpy(fnamebuf,argv[1]);

 fp = fopen(fnamebuf,"r");
 if(fp == NULL)
 {
   printf("Cant open file\n");
   exit(0);
 }

 fseek(fp, 0, SEEK_END); // to set the file pointer to end of file to count the number of bytes
 size = ftell(fp); // to get size of file in bytes
 printf("Size of file:%d bytes\n",size);

 nop = size/packetsize + (size%packetsize != 0); // to get ceil of quotient
 printf("Number of packets needed:%d\n",nop);

 // if(nop < 8) window_size = nop - 1;
 // else if(nop > 8 && nop < 16) window_size = 8;
 // else window_size = 16;

 //maximum_sequence_number = window_size - 1;

 fseek(fp, 0, SEEK_SET); // to set the file pointer to beginning of file to read contents
 buf = malloc(size);
 if(buf)
 {
  fread(buf,1,size,fp);
 }
 buf[size] = '\0';

 fclose(fp);
 /*** End of file operations ***/
```

```
//printf("%s\n",buf);

char str[30];
int numpackets = nop;
int msn = maximum_sequence_number;
int sw[window_size];
int packetarray[nop];
int i = 0;
int rem;

for(i=0;i<nop;i++)
{
  if(i <= msn)
  {
    if(i==0) packetarray[i] = 0;
    else if(i == msn) packetarray[i] = msn;
    else packetarray[i] = i % msn;
  }
  else packetarray[i] = i % (msn + 1);
  //printf("PACKARR:%d\n",packetarray[i]);
}

for(i=0;i<window_size;i++)
{
  sw[i] = packetarray[i];
}

int j;
int packindex = 0;
char packetstring[packetsize+1];
char abc[nop][packetsize+1];
int packnum = 0;
for(j=0;j<size;j++)
{
 packetstring[packindex] = buf[j];
 if(packindex == packetsize-1)
 {
   packetstring[packetsize] = '\0';
   strcpy(abc[packnum],packetstring);
   packnum++;
   packindex = 0;
 }
 else if(j == size-1)
 {
   packindex++;
   packetstring[packindex] = '\0';
   strcpy(abc[packnum],packetstring);
 }
 else
 {
   packindex++;
 }
}

free(buf);
```

```
gettimeofday(&start, NULL);

printf("\nSITUATIONAL ERRORS\n");
printf("------------------\n");
printf("1.None\n");
printf("2.Packet damaged (Random)\n");
printf("3.Packet lost (Random)\n");
printf("4.Ack lost (Random)\n");
printf("5.Packet damaged (User input)\n");
printf("6.Packet lost (User input)\n");
printf("7.Ack lost (User input)\n");
printf("Enter your choice:");
scanf("%d",&errchoice);
switch(errchoice)
  {
    case 1:perfectimp(numpackets, msn, window_size, sock, packetarray, abc, sw);
          break;

    case 2:damagedpacket(numpackets, msn, window_size, sock, packetarray, abc, sw, 1); // 2 => 3rd
packet
          break;

    case 3:lostpacket(numpackets, msn, window_size, sock, packetarray, abc, sw, 1);
          break;

    case 4:lostacknowledgement(numpackets, msn, window_size, sock, packetarray, abc, sw, 1);
          break;

    case 5:printf("Choose packet:");
          scanf("%d",&uichoice);
          damagedpacket(numpackets, msn, window_size, sock, packetarray, abc, sw, uichoice); // 2 =>
3rd packet
          break;

    case 6:printf("Choose packet:");
          scanf("%d",&uichoice);
          lostpacket(numpackets, msn, window_size, sock, packetarray, abc, sw, uichoice);
          break;

    case 7:printf("Choose packet:");
          scanf("%d",&uichoice);
          lostacknowledgement(numpackets, msn, window_size, sock, packetarray, abc, sw, uichoice);
          break;

    default:printf("Invalid choice\n");
           break;
  }

gettimeofday(&end, NULL);

double etime = (end.tv_sec * 1000000 + end.tv_usec) - (start.tv_sec * 1000000 + start.tv_usec);
double simtime = etime/1000000;
double throughput = size/simtime;
printf("Effective throughput:%0.02fBps\n",throughput);
printf("Simulation time:%0.02fs\n",simtime);

close(sock);
}
```

RECEIVER (receiver.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>

void printrw(int expectedpacket)
{
  printf("Current window = [");
  printf("%d",expectedpacket);
  printf("]\n");
}

void invertbits(char *a, char *b)
{
  int len = strlen(a);
  int i;
  for(i=0;i<len;i++)
  {
    if(a[i] == '0') b[i] = 'f';
    else if(a[i] == '1') b[i] = 'e';
    else if(a[i] == '2') b[i] = 'd';
    else if(a[i] == '3') b[i] = 'c';
    else if(a[i] == '4') b[i] = 'b';
    else if(a[i] == '5') b[i] = 'a';
    else if(a[i] == '6') b[i] = '9';
    else if(a[i] == '7') b[i] = '8';
    else if(a[i] == '8') b[i] = '7';
    else if(a[i] == '9') b[i] = '6';
    else if(a[i] == 'a') b[i] = '5';
    else if(a[i] == 'b') b[i] = '4';
    else if(a[i] == 'c') b[i] = '3';
    else if(a[i] == 'd') b[i] = '2';
    else if(a[i] == 'e') b[i] = '1';
    else b[i] = '0';
  }
}

int checksumverify(char *a, char *b, int sumlen)
{
   char *recsum;
   char *reccheck;
   char *invertedrecsum;
   int chk;
   //printf("%s,%s,%d\n",a,b,sumlen);
   recsum = malloc(sumlen);
   reccheck = malloc(sumlen);
   invertedrecsum = malloc(sumlen);
   recsum = a;
   reccheck = b;
   invertbits(recsum,invertedrecsum);
   if(strcmp(invertedrecsum,reccheck) != 0)
   {
     printf("Checksum failed\n");
     chk = 1;
   }
   else
   {
     printf("Checksum OK\n");
     chk = 0;
   }
   free(reccheck); // this frees memory occupied by reccheck and checksum since they occupy same
location
   free(invertedrecsum);
   free(recsum);
   return chk;
}

void unpack(char *a, int strlength, FILE *fp)
{
  int len = strlen(a);
  //printf("LEN%d\n",len);
  //printf("%s\n",a);
  char ch;
  char *opstring = (char*) malloc(strlength+1);
  opstring[0] = '\0';
  char *t; // dummy variable to use in strtol function
  char buf[3];
  char chbuf[2]; //converting ascii char to string and adding '\0' to use in strcat function
  int i = 0;
  int j = 0;
  int val;
  while(1)
  {
   if(a[i] == '|') break;
   //if(i == len) break;
   //printf("Char:%c\n",a[i]);
   buf[j] = a[i];
   i++;
   j++;
   if(j==2)
   {
     buf[j] = '\0';
     j = 0;
     val = strtol(buf,&t,16); //integer equivalent of the hex value
     ch = val; // ascii character corresponding to integer value
     chbuf[0] = ch;
     chbuf[1] = '\0';
     strcat(opstring,chbuf);
   }
  }
  //printf("%s\n",opstring);
  fprintf(fp,"%s",opstring);
  free(opstring);
  free(a);
}

int main()
{
int sock = socket(AF_INET, SOCK_STREAM, 0);

struct sockaddr_in s_address;
s_address.sin_family = AF_INET;
s_address.sin_port = htons(51374);
s_address.sin_addr.s_addr = INADDR_ANY;
if (bind(sock, (struct sockaddr *) &s_address, sizeof(s_address)) < 0)
{
   printf("Bind error\n");
   exit(0);
}

FILE *fp;

fp = fopen("output.txt","w");

int tmlen; // length of received msg
char *receivedmsg; // the msg received
int sumlen; //length of received sum/checksum
char *sumbuf;
char *checksum;
int msglen;

char str[30];
int packid,numpackets;
int flag = 0;
int loopcount;
int rw;
int msn;
int nextpacketindex = 0; // variable to hold the index of the next packet in the packet array;
//int *packetarray;
int packetflag = 0; //variable to control the transmission of packet information like number of
packets and max seq number
int numpacks = 0; // variable to keep track of the number of packets received
int check;
int acklostindex;
int resendflag = 0;
int resendackid;

listen(sock, 3);

struct sockaddr_in c_address;
socklen_t c_length = sizeof(c_address);
int new_sock = accept(sock, (struct sockaddr *) &c_address, &c_length);

do
{
```

```
if(packetflag == 0)
{
  read(new_sock,&numpackets,sizeof(numpackets));
  read(new_sock,&msn,sizeof(msn));
  read(new_sock,&acklostindex,sizeof(acklostindex));
  packetflag = 1;
}

int packetarray[numpackets];

if(flag == 0)
{
  loopcount = numpackets;
  //packetarray = (int *)malloc(sizeof(numpackets));
  int i;
  int rem;
  for(i=0;i<numpackets;i++)
  {
    if(i <= msn)
    {
      if(i==0) packetarray[i] = 0;
      else if(i == msn) packetarray[i] = msn;
      else packetarray[i] = i % msn;
    }
    else packetarray[i] = i % (msn + 1);
  }
  rw = packetarray[nextpacketindex];
  printrw(rw);
  flag = 1;
}
read(new_sock,&packid,sizeof(packid)); //read packid eg.0,1,2
printf("Packet %d received\n",packid);
read(new_sock,&tmlen,sizeof(tmlen));
//printf("%d\n",tmlen);
receivedmsg = malloc(tmlen+1);
read(new_sock,receivedmsg,tmlen+1);
//printf("%s\n",receivedmsg);
//printf("%d\n",strlen(receivedmsg));
read(new_sock,&sumlen,sizeof(sumlen));
sumbuf = malloc(sumlen+1);
checksum = malloc(sumlen+1);
read(new_sock,sumbuf,sumlen+1);
//printf("%s\n",sumbuf);
read(new_sock,checksum,sumlen+1);
read(new_sock,&msglen,sizeof(msglen));
if(packid == rw)
{
  check = checksumverify(sumbuf,checksum,sumlen);
}
else
{
  printf("Packet %d discarded\n",packid);
  check = 1;
}
if(check == 0)
```

```
  {
    if(numpacks != acklostindex || resendflag == 1)
    {
      if(resendflag == 1)
      {
        write(new_sock,&resendackid,sizeof(resendackid));
        numpacks++;
        resendflag = 0;
      }

        unpack(receivedmsg,msglen,fp);
        numpacks++;
        //printf("NUMPACKS:%d\n",numpacks);
        printf("Ack %d sent\n",packid);
        if(numpacks < numpackets)
        {
          nextpacketindex++;
          rw = packetarray[nextpacketindex];
          printrw(rw);
        }
        write(new_sock,&packid,sizeof(packid));
    }
    else
    {
      unpack(receivedmsg,msglen,fp);
      printf("Ack %d sent\n",packid);
      if(numpacks < numpackets)
      {
        nextpacketindex++;
        rw = packetarray[nextpacketindex];
        printrw(rw);
      }
      resendflag = 1;
      resendackid = packid;
    }
  }
  else
  {
     printrw(rw);
    //write(new_sock,&packid,sizeof(packid));
  }

}while(numpacks < numpackets);

//free(packetarray);

fclose(fp);

close(sock);
close(new_sock);
}
```