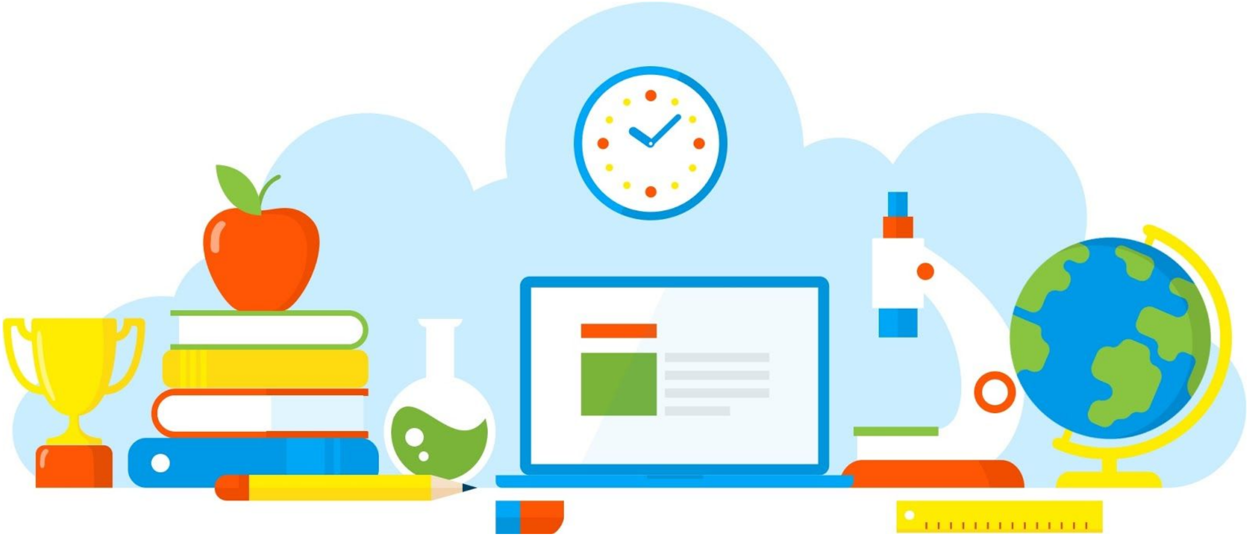


Assignment Report

*This report contains work from **Assignment 7** of course CSN 361.*

Name- Kaustubh Trivedi
Enrollment Number - 17114044
Class - CSE B.Tech. 3rd Year
Submission Files - [Repository Link]

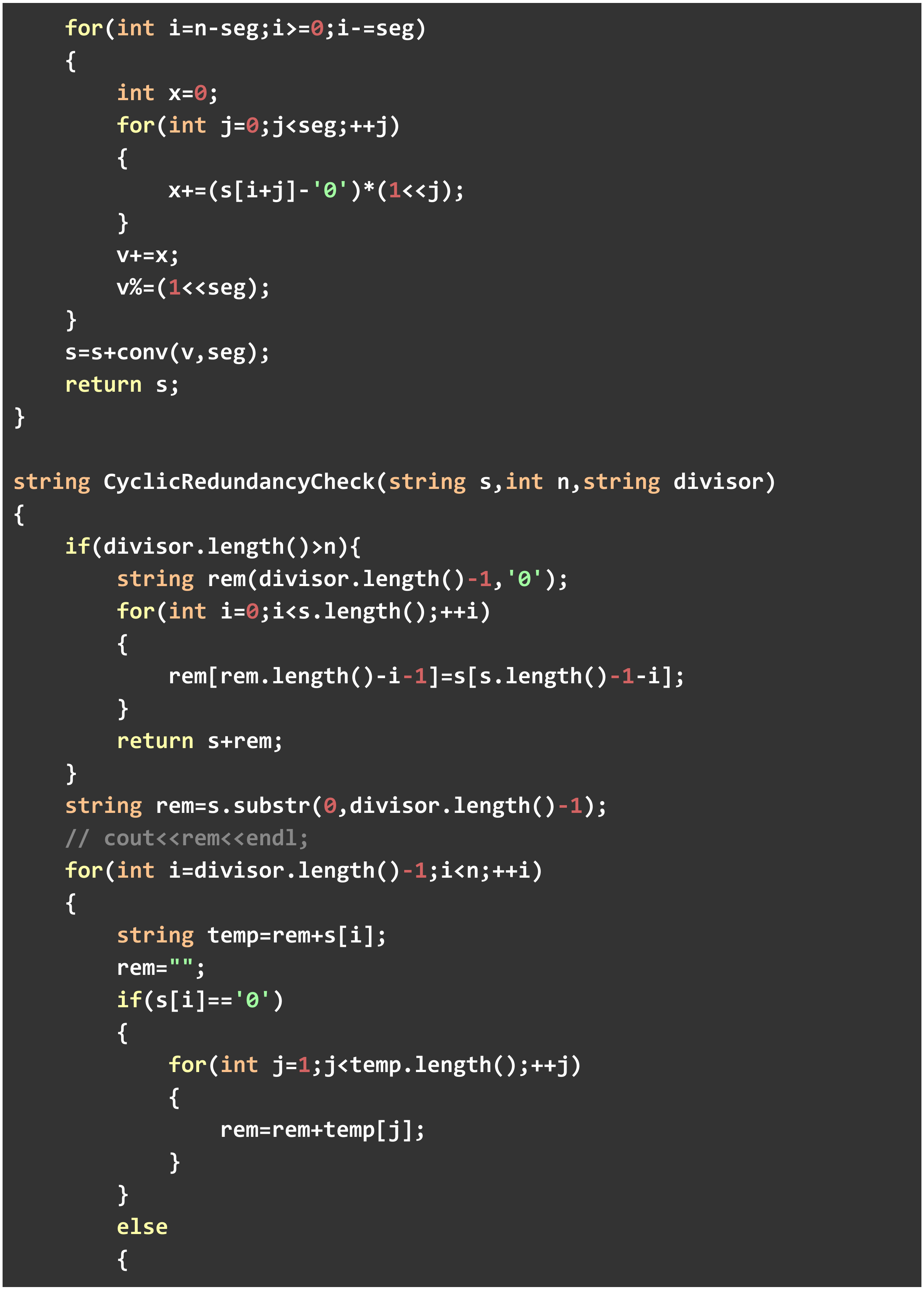
CLIENT CODE FOR QUESTION 1:

```
#include<bits/stdc++.h>
using namespace std;
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0
#define nofile "File Not Found!"

// function to clear buffer
void clearBuf(char* b)
{
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}

// function to encrypt
char Cipher(char ch)
{
    return ch ^ cipherKey;
}

// function sending file
int sendFile(FILE* fp, char* buf, int s)
{
    int i, len;
    if (fp == NULL) {
        strcpy(buf, nofile);
        len = strlen(nofile);
        buf[len] = EOF;
        for (i = 0; i <= len; i++)
            buf[i] = Cipher(buf[i]);
        return 1;
    }

    char ch, ch2;
    for (i = 0; i < s; i++) {
        ch = fgetc(fp);
        ch2 = Cipher(ch);
        buf[i] = ch2;
        if (ch == EOF)
            return 1;
    }
    return 0;
}


string CyclicRedundancyCheck(string s,int n,string divisor)
{
    if(divisor.length()>n){
        string rem(divisor.length()-1,'0');
        for(int i=0;i<s.length();++i)
        {
            rem[rem.length()-i-1]=s[s.length()-1-i];
        }
        return s+rem;
    }
    cout<<s<<" "<<divisor<<endl;
    string rem=s.substr(0,divisor.length()-1);
    // cout<<rem<<endl;
    for(int i=divisor.length()-1;i<n;++i)
    {
        string temp=rem+s[i];
        rem="";
        cout<<temp<<endl;
        if(temp[0]=='0')
        {
            for(int j=1;j<temp.length();++j)
            {
                rem=rem+temp[j];
            }
        }
        else
        {
            for(int j=1;j<temp.length();++j)
            {
                if(divisor[j]=='0')
                    rem=rem+temp[j];
                else{
                    if(temp[j]=='0')
                        rem=rem+"1";
                    else
                        rem=rem+"1";
                }
            }
        }
    }
    return s+rem;
}


int checksum(string s,int n,int seg)
{
    int v=0;
    for(int i=n-seg;i>=0;i-=seg)
    {
        int x=0;
        for(int j=0;j<seg;++j)
        {
            x+=(s[i+j]-'0')*(1<<j);
        }
        v+=x;
        v%=(1<<seg);
    }
    cout<<v<<endl;
    if(v==0)
        return 0;
    else
        return 1;
}
// driver code
int main()
{
    int sockfd, nBytes;
    struct sockaddr_in addr_con;
    unsigned int addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = INADDR_ANY;
    char net_buf[NET_BUF_SIZE];
    FILE* fp;

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

    if (sockfd < 0)
        printf("\nfile descriptor not received!!\n");
    else
        printf("\nfile descriptor %d received\n", sockfd);

    // bind()
    if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con))
== 0)
        printf("\nSuccessfully binded!\n");
    else
        printf("\nBinding Failed!\n");

    while (1) {
        printf("\nWaiting for string\n");

        clearBuf(net_buf);

        int ty;
        nBytes = recvfrom(sockfd, net_buf,
                          NET_BUF_SIZE, sendrecvflag,
                          (struct sockaddr*)&addr_con, &addrlen);

        ty = stoi(string(net_buf));
        int n;

        clearBuf(net_buf);
        nBytes = recvfrom(sockfd, net_buf,
                          NET_BUF_SIZE, sendrecvflag,
                          (struct sockaddr*)&addr_con, &addrlen);

        n = stoi(string(net_buf));

        clearBuf(net_buf);
        int error=0;
        string s;
        switch(ty)
        {
            case 1:
            {
                cout<<"Checker Type Single Parity Checker\n";
                clearBuf(net_buf);
                nBytes = recvfrom(sockfd, net_buf,
                                  NET_BUF_SIZE, sendrecvflag,
                                  (struct sockaddr*)&addr_con,
&addrlen);
                s=string(net_buf);
                int v=0;
                for(int i=0;i<s.length()-1;++i)
                {
                    if(s[i]=='1')
                        v=!v;
                }
                if(v)
                {
                    if(s[s.length()-1]=='0')
                        error=1;
                }
                else
                {
                    if(s[s.length()-1]=='1')
                        error=1;
                }
            }
            break;
            case 2:
            {
                cout<<"Checker Type Two-dimensional Parity Check\n";
                clearBuf(net_buf);
                nBytes = recvfrom(sockfd, net_buf,
                                  NET_BUF_SIZE, sendrecvflag,
                                  (struct sockaddr*)&addr_con,
&addrlen);

                int seg = stoi(string(net_buf));
                int mat[n/seg][seg];
                int a=n/seg;
                int b=seg;

                clearBuf(net_buf);
                nBytes = recvfrom(sockfd, net_buf,
                                  NET_BUF_SIZE, sendrecvflag,
                                  (struct sockaddr*)&addr_con,
&addrlen);
                s=string(net_buf);

                for(int i=0;i<n;++i)
                {
                    if(s[i]=='1')
                        mat[i/seg][i%seg]=1;
                    else
                        mat[i/seg][i%seg]=0;
                }
                int z=0;
                for(int i=0;i<a;++i)
                {
                    int v=0;
                    for(int j=0;j<b;++j){
                        cout<<mat[i][j];
                        v^=mat[i][j];
                    }
                    cout<<"\n";
                    if(s[n+z]!=v+'0')
                        error=1;
                        ++z;
                }
                for(int i=0;i<b;++i)
                {
                    int v=0;
                    for(int j=0;j<a;++j)
                        v^=mat[j][i];
                    if(s[n+z]!=v+'0')
                        error=1;
                        ++z;
                }
            }
            break;
            case 3:
            {
                cout<<"Checker Type Checksum\n";
                clearBuf(net_buf);
                nBytes = recvfrom(sockfd, net_buf,
                                  NET_BUF_SIZE, sendrecvflag,
                                  (struct sockaddr*)&addr_con,
&addrlen);

                int seg = stoi(string(net_buf));
                clearBuf(net_buf);
                nBytes = recvfrom(sockfd, net_buf,
                                  NET_BUF_SIZE, sendrecvflag,
                                  (struct sockaddr*)&addr_con,
&addrlen);
                s=string(net_buf);
                int val=checksum(s,s.length(),seg);
                if(val==1)
                {
                    error=1;
                }
                else
                    error=0;
            }
            break;
            case 4:
            {
                cout<<"Checker Type Cyclic Redundancy Check(CRC)\n";
                clearBuf(net_buf);
                nBytes = recvfrom(sockfd, net_buf,
                                  NET_BUF_SIZE, sendrecvflag,
                                  (struct sockaddr*)&addr_con,
&addrlen);

                string divisor = string(net_buf);
                clearBuf(net_buf);
                nBytes = recvfrom(sockfd, net_buf,
                                  NET_BUF_SIZE, sendrecvflag,
                                  (struct sockaddr*)&addr_con,
&addrlen);
                s=string(net_buf);
                string
val=CyclicRedundancyCheck(s.substr(0,s.length()-divisor.length()+1),s
.length()-divisor.length()+1,divisor);
                cout<<val<<endl;
                if(val==s)
                {
                    error=0;
                }
                else
                    error=1;
            }
            break;
        }
        printf("\nString Received: %s\n", s.c_str());

        if(error==0)
        {
            cout<<"NO ERROR DETECTED\n";
        }
        else
        {
            cout<<"ERROR DETECTED\n";
        }

    }
    return 0;
}
```

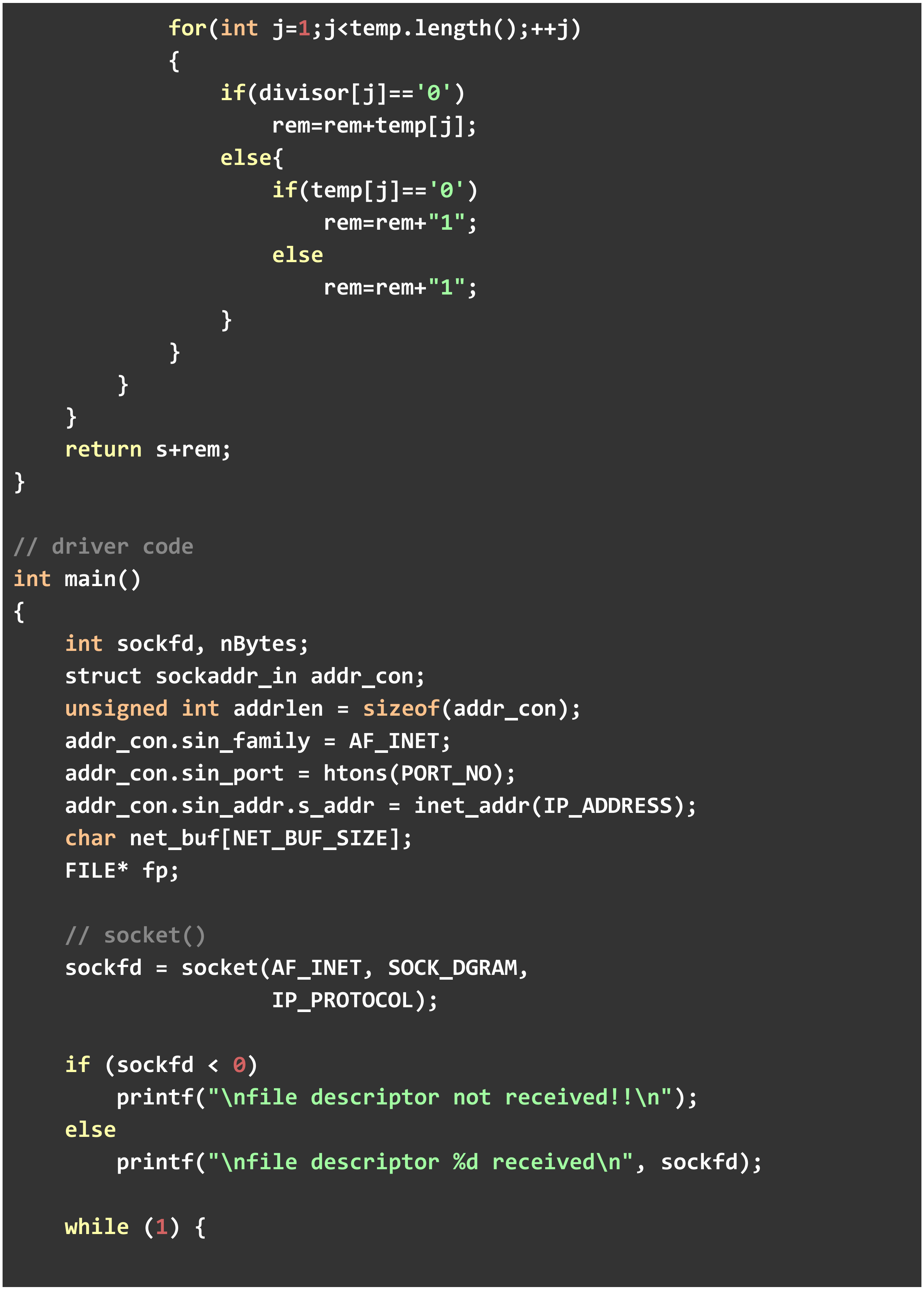
SERVER CODE FOR QUESTION 1:

```
#include<bits/stdc++.h>
using namespace std;
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define IP_ADDRESS "127.0.0.1" // localhost
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0

// function to clear buffer
void clearBuf(char* b)
{
    int i;
    for (i = 0; i < NET_BUF_SIZE; i++)
        b[i] = '\0';
}
```

```
// function for decryption
char Cipher(char ch)
{
    return ch ^ cipherKey;
}

// function to receive file
int recvFile(char* buf, int s)
{
    int i;
    char ch;
    for (i = 0; i < s; i++) {
        ch = buf[i];
        ch = Cipher(ch);
        if (ch == EOF)
            return 1;
        else
            printf("%c", ch);
    }
    return 0;
}

string conv(int val,int seg)
{
    string s=string(seg,'1');
    for(int i=0;i<seg;++i)
    {
        if(val&(1<<(seg-i-1)))
        {
            s[i]='0';
        }
    }
    return s;
}

string checksum(string s,int n,int seg)
{
    int v=0;
```

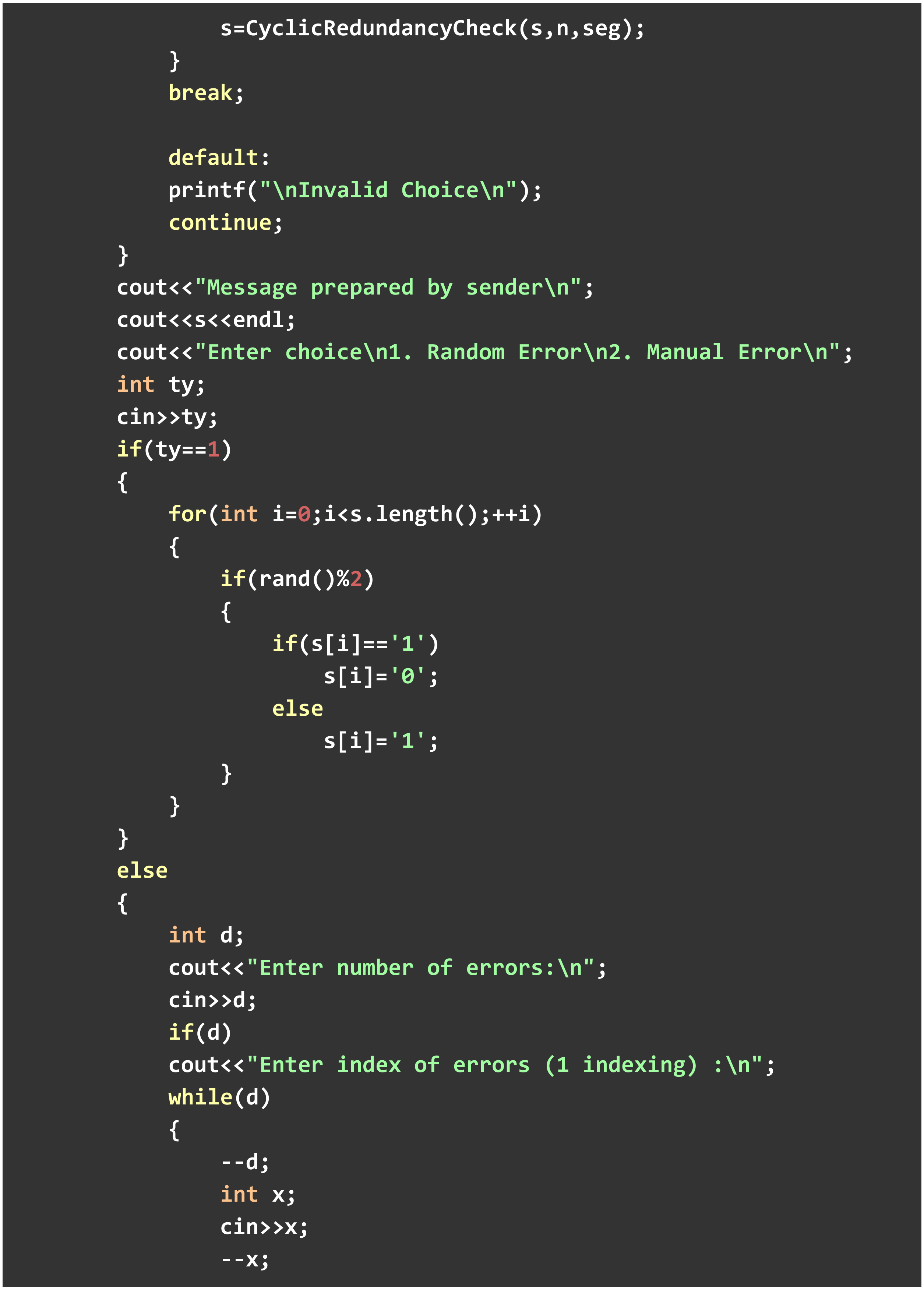


```
    for(int i=n-seg;i>=0;i-=seg)
    {
        int x=0;
        for(int j=0;j<seg;++j)
        {
            x+=(s[i+j]-'0')*(1<<j);
        }
        v+=x;
        v%=(1<<seg);
    }
    s=s+conv(v,seg);
    return s;
}

string CyclicRedundancyCheck(string s,int n,string divisor)
{
    if(divisor.length()>n){
        string rem(divisor.length()-1,'0');
        for(int i=0;i<s.length();++i)
        {
            rem[rem.length()-i-1]=s[s.length()-1-i];
        }
        return s+rem;
    }
    string rem=s.substr(0,divisor.length()-1);
    // cout<<rem<<endl;
    for(int i=divisor.length()-1;i<n;++i)
    {
        string temp=rem+s[i];
        rem="";
        if(s[i]=='0')
        {
            for(int j=1;j<temp.length();++j)
            {
                rem=rem+temp[j];
            }
        }
        else
        {
```



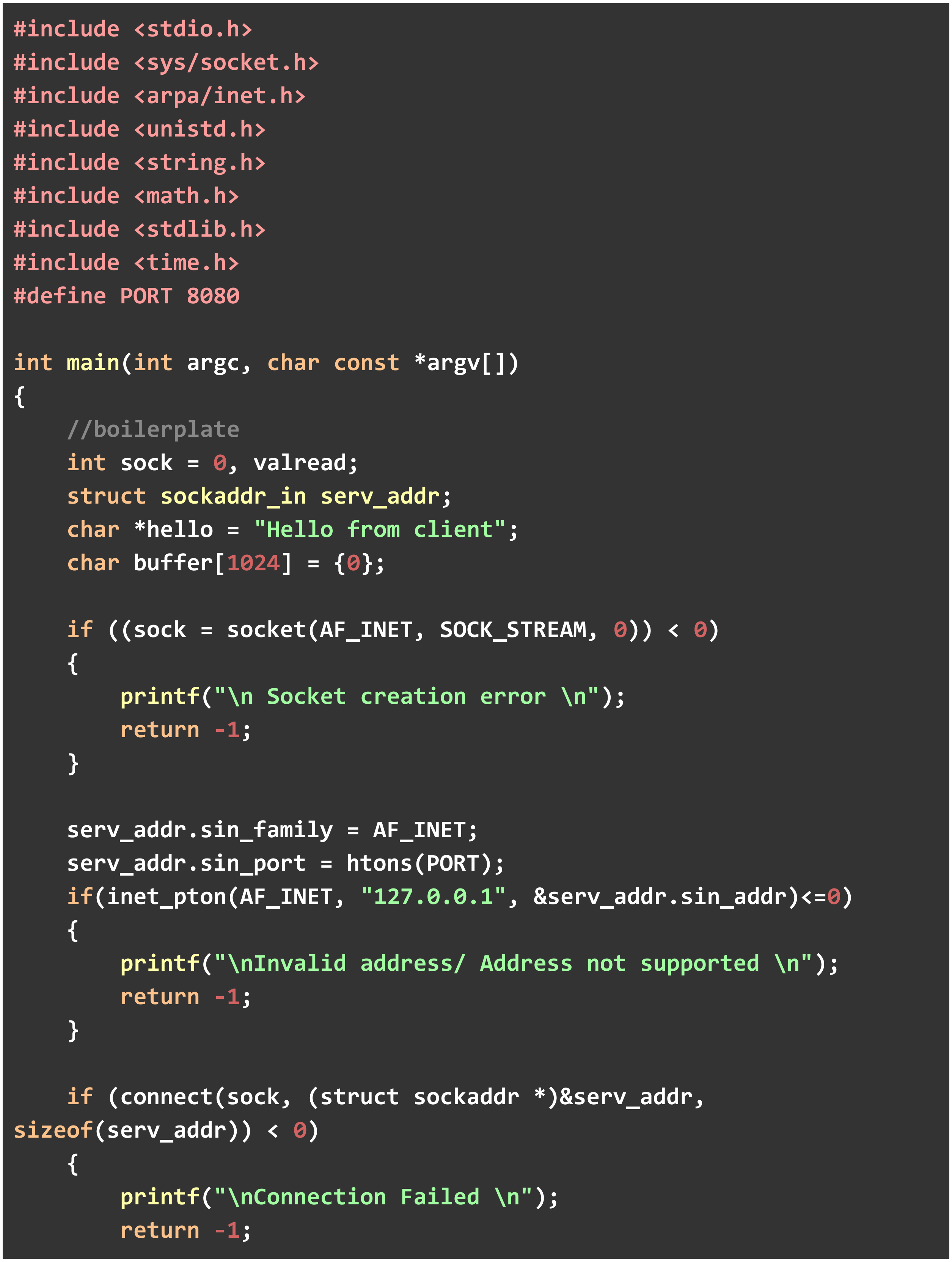
```
            for(int j=1;j<temp.length();++j)
            {
                if(divisor[j]=='0')
                    rem=rem+temp[j];
                else{
                    if(temp[j]=='0')
                        rem=rem+"1";
                    else
                        rem=rem+"1";
                }
            }
        }
    }
    return s+rem;
}

// driver code
int main()
{
    int sockfd, nBytes;
    struct sockaddr_in addr_con;
    unsigned int addrlen = sizeof(addr_con);
    addr_con.sin_family = AF_INET;
    addr_con.sin_port = htons(PORT_NO);
    addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
    char net_buf[NET_BUF_SIZE];
    FILE* fp;

    // socket()
    sockfd = socket(AF_INET, SOCK_DGRAM,
                    IP_PROTOCOL);

    if (sockfd < 0)
        printf("\nfile descriptor not received!!\n");
    else
        printf("\nfile descriptor %d received\n", sockfd);

    while (1) {
        string s;

        printf("\nSelect the algorithm \n1.Single Parity Check \n2.Two-dimensional Parity Check\n3. Checksum \n4. Cyclic Redundancy Check(CRC)\n");

        int choice;
        scanf("%d",&choice);

        strcpy(net_buf,to_string(choice).c_str());

        sendto(sockfd, net_buf, NET_BUF_SIZE,
               sendrecvflag, (struct sockaddr*)&addr_con,
               addrlen);

        int n;
        cout<<"Enter length of message :\n";
        cin>>n;

        s=to_string(n);
        strcpy(net_buf,s.c_str());

        sendto(sockfd, net_buf, NET_BUF_SIZE,
               sendrecvflag, (struct sockaddr*)&addr_con,
               addrlen);

        cout<<"Enter message :\n";
        cin>>s;
        switch(choice)
        {
            case 1:
            {
                int v=0;
                for(int i=0;i<s.length();++i)
                {
                    char r=s[i];
                    if(r=='1')
                        v=!v;
                }
                if(v)
                    s=s+"1";
                else
                    s=s+"0";
            }
            break;
            case 2:
            {
                cout<<"Enter length of segments \n";
                int seg;
                cin>>seg;
                string ss=to_string(seg);
                strcpy(net_buf,ss.c_str());

                sendto(sockfd, net_buf, NET_BUF_SIZE,
                       sendrecvflag, (struct sockaddr*)&addr_con,
                       addrlen);
                int mat[n/seg][seg];
                int a=n/seg;
                int b=seg;
                for(int i=0;i<n;++i)
                {
                    if(s[i]=='1')
                        mat[i/seg][i%seg]=1;
                    else
                        mat[i/seg][i%seg]=0;
                }
                cout<<"Matrix prepared\n";
                for(int i=0;i<a;++i)
                {
                    int v=0;
                    for(int j=0;j<b;++j){
                        cout<<mat[i][j];
                        v^=mat[i][j];
                    }
                    cout<<"\n";
                    if(v==1)
                        s=s+"1";
                    else
```

```
                        s=s+"0";
                }
                for(int i=0;i<b;++i)
                {
                    int v=0;
                    for(int j=0;j<a;++j)
                        v^=mat[j][i];
                    if(v==1)
                        s=s+"1";
                    else
                        s=s+"0";
                }
            }
            break;
            case 3:
            {
                cout<<"Enter length of segment \n";
                int seg;
                cin>>seg;
                string ss=to_string(seg);
                strcpy(net_buf,ss.c_str());

                sendto(sockfd, net_buf, NET_BUF_SIZE,
                       sendrecvflag, (struct sockaddr*)&addr_con,
                       addrlen);
                s=checksum(s,n,seg);
            }
            break;
            case 4:
            {
                cout<<"Enter divisor \n";
                string seg;
                cin>>seg;

                strcpy(net_buf,seg.c_str());

                sendto(sockfd, net_buf, NET_BUF_SIZE,
                       sendrecvflag, (struct sockaddr*)&addr_con,
                       addrlen);
```

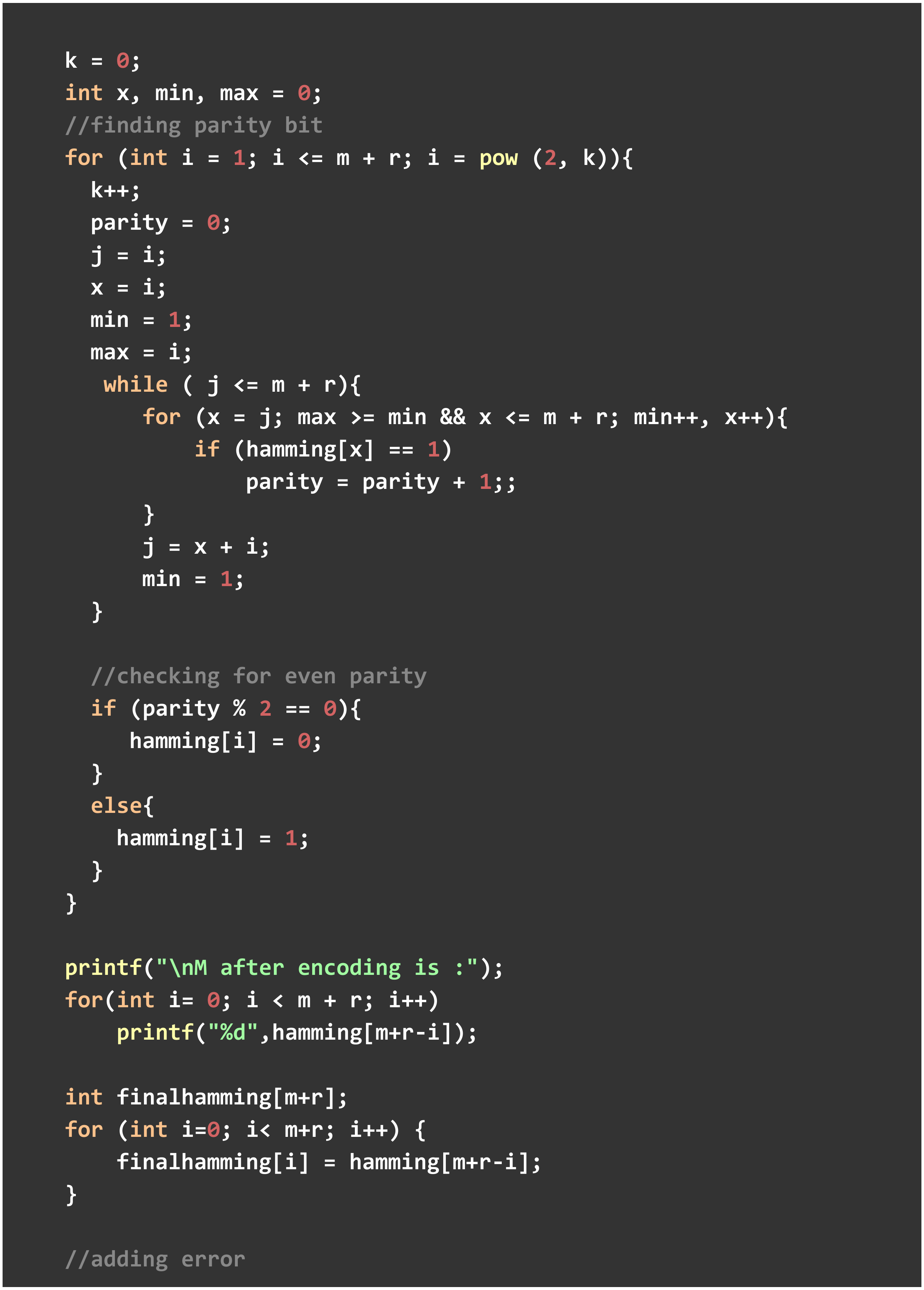


```
                s=CyclicRedundancyCheck(s,n,seg);
            }
            break;

            default:
            printf("\nInvalid Choice\n");
            continue;
        }
        cout<<"Message prepared by sender\n";
        cout<<s<<endl;
        cout<<"Enter choice\n1. Random Error\n2. Manual Error\n";
        int ty;
        cin>>ty;
        if(ty==1)
        {
            for(int i=0;i<s.length();++i)
            {
                if(rand()%2)
                {
                    if(s[i]=='1')
                        s[i]='0';
                    else
                        s[i]='1';
                }
            }
        }
        else
        {
            int d;
            cout<<"Enter number of errors:\n";
            cin>>d;
            if(d)
            cout<<"Enter index of errors (1 indexing) :\n";
            while(d)
            {
                --d;
                int x;
                cin>>x;
                --x;
                if(s[x]=='1')
                    s[x]='0';
                else
                    s[x]='1';
            }
        }
        strcpy(net_buf,s.c_str());

        sendto(sockfd, net_buf, NET_BUF_SIZE,
               sendrecvflag, (struct sockaddr*)&addr_con,
               addrlen);

        printf("\n---------Data Sent---------\n");

        printf("%s", net_buf );

        printf("\n-------------------------------\n");

    }
    return 0;
}
```

SERVER CODE FOR QUESTION 2:



```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#define PORT 8080

int main(int argc, char const *argv[])
{
    //boilerplate
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char *hello = "Hello from client";
    char buffer[1024] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }




    //end boilerplate
    int m, data_bits[20],r = 0,parity;    //m = no. of data bits, r = no. of redundant bits

    //input number of bits
    printf("Enter the number of bits: ");
    scanf("%d", &m);

    //input number
    printf("Enter the N bit message, bit by bit: \n");
    for(int i = 0; i < m; i++)
        scanf("%d", &data_bits[m-i]);

    //finding redundant bits
    while(pow (2,r) < m + r + 1){
        r++;
    }
    printf("Redundant Bits: %d", r);

    int hamming[m + r + 1],j = 0,k = 1;

    //finding positions of redundant bits.
    for(int i = 1; i <= m + r; i++){

        if( i == pow( 2, j )){
            hamming[i] = -1;    //-1 is initial value of redundant bits
            j++;
        }
        else{
            hamming[i] = data_bits[k];
            k++;
        }
    }
```

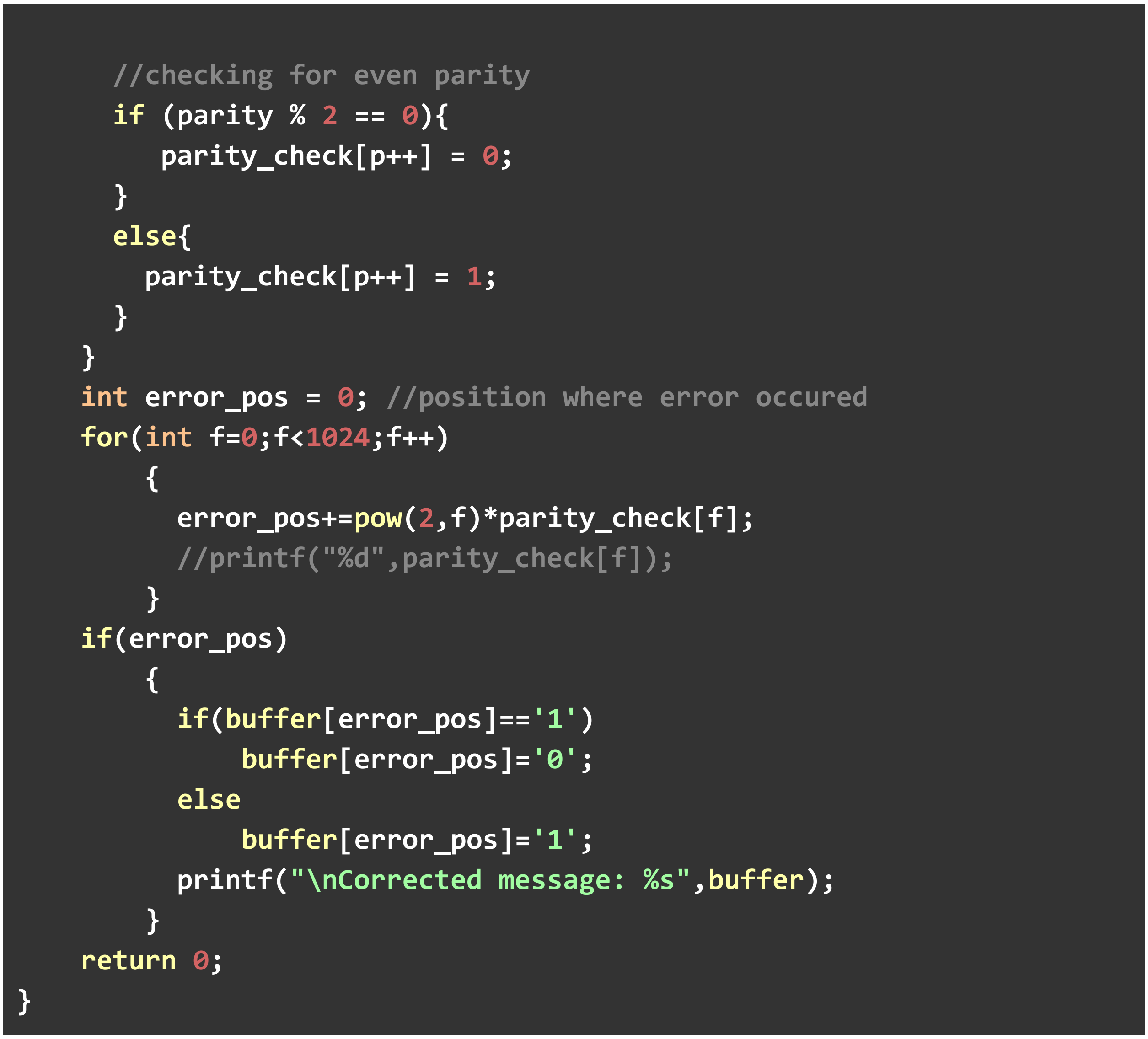


```
k = 0;
int x, min, max = 0;
//finding parity bit
for (int i = 1; i <= m + r; i = pow (2, k)){
  k++;
  parity = 0;
  j = i;
  x = i;
  min = 1;
  max = i;
   while ( j <= m + r){
      for (x = j; max >= min && x <= m + r; min++, x++){
          if (hamming[x] == 1)
              parity = parity + 1;;
      }
      j = x + i;
      min = 1;
  }

  //checking for even parity
  if (parity % 2 == 0){
     hamming[i] = 0;
  }
  else{
    hamming[i] = 1;
  }
}

printf("\nM after encoding is :");
for(int i= 0; i < m + r; i++)
    printf("%d",hamming[m+r-i]);

int finalhamming[m+r];
for (int i=0; i< m+r; i++) {
    finalhamming[i] = hamming[m+r-i];
}

//adding error
```

```
    printf("\nDo you wish to add an error?");
    printf("\nPress 0 for Manual error");
    printf("\nPress 1 for random errors");
    printf("\nPress any other key for no errors\n");

    int res;
    scanf("%d", &res);

    if (res==0){
        printf("\nPress the number of bits to be flipped :");
        int num;
        scanf("%d", &num);
        printf("\nEnter the indices of the bits to be flipped\n");
        for (int i =0; i< num; i++){
            int pos;
            scanf("%d", &pos);
            finalhamming[pos] = (finalhamming[pos]+1)%2;
        }
    }
    if (res==1){
        int index= rand()%(m+r);
        printf("\nBit%d is flipped\n", index);

        finalhamming[index] = (finalhamming[index]+1)%2;
    }
    printf("\nM transmitted (after adding errors): ");
    for(int i= 0; i < m + r; i++)
        printf("%d",finalhamming[i]);
    char message[m+r];
    for(int i= 0; i < m + r; i++){
        message[i]=(finalhamming[i]==0)?'0':'1';
    }


    send(sock , message , m+r , 0 );
    printf("\nEncoded message sent\n");

    return 0;
}
```

CLIENT CODE FOR QUESTION 2:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <math.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR |
SO_REUSEPORT,
                                                  &opt, sizeof(opt)))
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons( PORT );

    if (bind(server_fd, (struct sockaddr *)&address,
                                 sizeof(address))<0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                       (socklen_t*)&addrlen))<0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    printf("Listening for encoded message");
    valread = read( new_socket , buffer, 1024);
    printf("\nMessage received is : %s\n",buffer );

    //int size = buffer.size()-1;
    int size = valread;
    int parity_check[1024] = {0};
    // Checking error
    int j,x,min,max,p=0;
    int k=0;
    for (int i = 1; i <= size; i = pow (2, k)){
      k++;
      int parity = 0;
      j = i;
      x = i;
      min = 1;
      max = i;
       while ( j <= size){
          for (x = j; max >= min && x <= size; min++, x++){
              if (buffer[x] == '1')
                  parity = parity + 1;
              //printf("%d",x);
          }
          j = x + i;
          min = 1;
      }
```



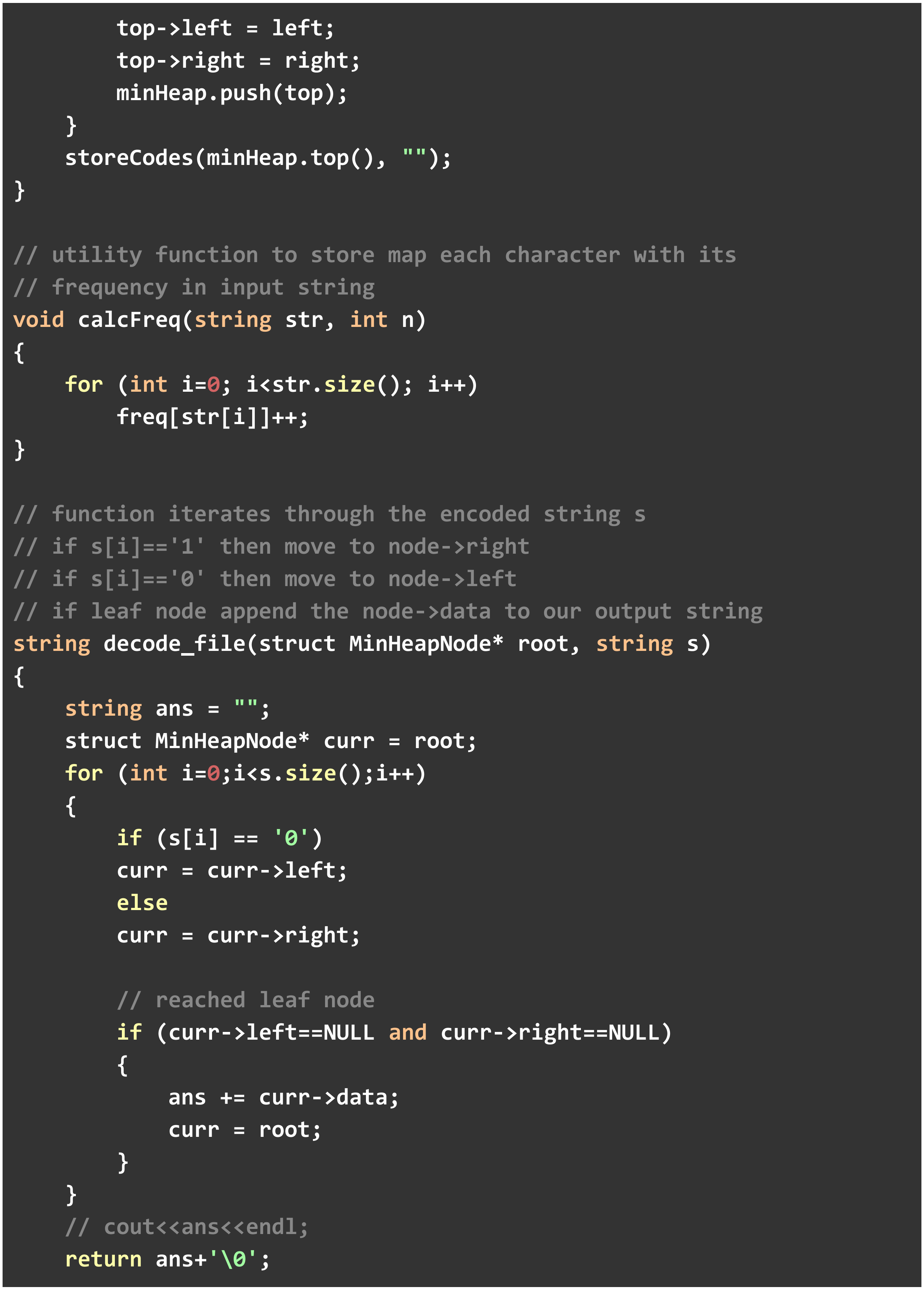
```
      //checking for even parity
      if (parity % 2 == 0){
         parity_check[p++] = 0;
      }
      else{
        parity_check[p++] = 1;
      }
    }
    int error_pos = 0; //position where error occured
    for(int f=0;f<1024;f++)
        {
          error_pos+=pow(2,f)*parity_check[f];
          //printf("%d",parity_check[f]);
        }
    if(error_pos)
        {
          if(buffer[error_pos]=='1')
              buffer[error_pos]='0';
          else
              buffer[error_pos]='1';
          printf("\nCorrected message: %s",buffer);
        }
    return 0;
}
```

SOLUTION CODE FOR QUESTION 3:

```
#include <bits/stdc++.h>
#include <string>
#include <fstream>
#include <streambuf>


#define MAX_TREE_HT 256

using namespace std;

// Shannon Fano
struct node {

    // for storing symbol
    string sym;

    // for storing probability or frquency
    float pro;
    int arr[20];
    int top;
};

struct node *p;
typedef struct node node;

map <string, string> shannonCodes;
map <string,string> shannonRev;
// function to find shannon code
void shannon(int l, int h, node p[])
{
    float pack1 = 0, pack2 = 0, diff1 = 0, diff2 = 0;
    int i, d, k, j;
    if ((l + 1) == h || l == h || l > h) {
        if (l == h || l > h)
            return;
        p[h].arr[++(p[h].top)] = 0;
        p[l].arr[++(p[l].top)] = 1;
        return;
    }
    else {
        for (i = l; i <= h - 1; i++)
            pack1 = pack1 + p[i].pro;
        pack2 = pack2 + p[h].pro;
        diff1 = pack1 - pack2;
        if (diff1 < 0)
            diff1 = diff1 * -1;
        j = 2;
        while (j != h - l + 1) {
            k = h - j;
            pack1 = pack2 = 0;
            for (i = l; i <= k; i++)
                pack1 = pack1 + p[i].pro;
            for (i = h; i > k; i--)
                pack2 = pack2 + p[i].pro;
            diff2 = pack1 - pack2;
            if (diff2 < 0)
                diff2 = diff2 * -1;
            if (diff2 >= diff1)
                break;
            diff1 = diff2;
            j++;
        }
        k++;
        for (i = l; i <= k; i++)
            p[i].arr[++(p[i].top)] = 1;
        for (i = k + 1; i <= h; i++)
            p[i].arr[++(p[i].top)] = 0;

        // Invoke shannon function
        shannon(l, k, p);
        shannon(k + 1, h, p);
    }
}

// Function to sort the symbols
// based on their probability or frequency
```

```
void sortByProbability(int n, node p[])
{
    int i, j;
    node temp;
    for (j = 1; j <= n - 1; j++) {
        for (i = 0; i < n - 1; i++) {
            if ((p[i].pro) > (p[i + 1].pro)) {
                temp.pro = p[i].pro;
                temp.sym = p[i].sym;

                p[i].pro = p[i + 1].pro;
                p[i].sym = p[i + 1].sym;

                p[i + 1].pro = temp.pro;
                p[i + 1].sym = temp.sym;
            }
        }
    }
}

// function to display shannon codes
void display(int n, node p[])
{
    int i, j;
    cout << "\n\n\n\tSymbol\tProbability\tCode";

    for (i = n - 1; i >= 0; i--) {
        string code = "";
        cout << "\n\t" << p[i].sym << "\t\t" << p[i].pro << "\t";
        for (j = 0; j <= p[i].top; j++) {
            code = code + to_string(p[i].arr[j]);
            cout << p[i].arr[j];
        }
        shannonCodes[code] = p[i].sym;
        shannonRev[p[i].sym] = code;

    }
    cout<<endl;
```

```
}




// Huffman
map<char, string> codes;

// to store the frequency of character of the input data
map<char, int> freq;

// A Huffman tree node
struct MinHeapNode
{
    char data;             // One of the input characters
    int freq;              // Frequency of the character
    MinHeapNode *left, *right; // Left and right child

    MinHeapNode(char data, int freq)
    {
        left = right = NULL;
        this->data = data;
        this->freq = freq;
    }
};

// utility function for the priority queue
struct compare
{
    bool operator()(MinHeapNode* l, MinHeapNode* r)
    {
        return (l->freq > r->freq);
    }
};

// utility function to print characters along with
// there huffman value
void printCodes(struct MinHeapNode* root, string str)
{
    if (!root)
        return;
    if (root->data != '$')
        cout << root->data << ": " << str << "\n";
    printCodes(root->left, str + "0");
    printCodes(root->right, str + "1");
}

// utility function to store characters along with
// there huffman value in a hash table, here we
// have C++ STL map
void storeCodes(struct MinHeapNode* root, string str)
{
    if (root==NULL)
        return;
    if (root->data != '$')
        codes[root->data]=str;
    storeCodes(root->left, str + "0");
    storeCodes(root->right, str + "1");
}

// STL priority queue to store heap tree, with respect
// to their heap root node value
priority_queue<MinHeapNode*, vector<MinHeapNode*>, compare> minHeap;

// function to build the Huffman tree and store it
// in minHeap
void HuffmanCodes(int size)
{
    struct MinHeapNode *left, *right, *top;
    for (map<char, int>::iterator v=freq.begin(); v!=freq.end(); v++)
        minHeap.push(new MinHeapNode(v->first, v->second));
    while (minHeap.size() != 1)
    {
        left = minHeap.top();
        minHeap.pop();
        right = minHeap.top();
        minHeap.pop();
        top = new MinHeapNode('$', left->freq + right->freq);
```



```
        top->left = left;
        top->right = right;
        minHeap.push(top);
    }
    storeCodes(minHeap.top(), "");
}

// utility function to store map each character with its
// frequency in input string
void calcFreq(string str, int n)
{
    for (int i=0; i<str.size(); i++)
        freq[str[i]]++;
}

// function iterates through the encoded string s
// if s[i]=='1' then move to node->right
// if s[i]=='0' then move to node->left
// if leaf node append the node->data to our output string
string decode_file(struct MinHeapNode* root, string s)
{
    string ans = "";
    struct MinHeapNode* curr = root;
    for (int i=0;i<s.size();i++)
    {
        if (s[i] == '0')
        curr = curr->left;
        else
        curr = curr->right;

        // reached leaf node
        if (curr->left==NULL and curr->right==NULL)
        {
            ans += curr->data;
            curr = root;
        }
    }
    // cout<<ans<<endl;
    return ans+'\0';
```

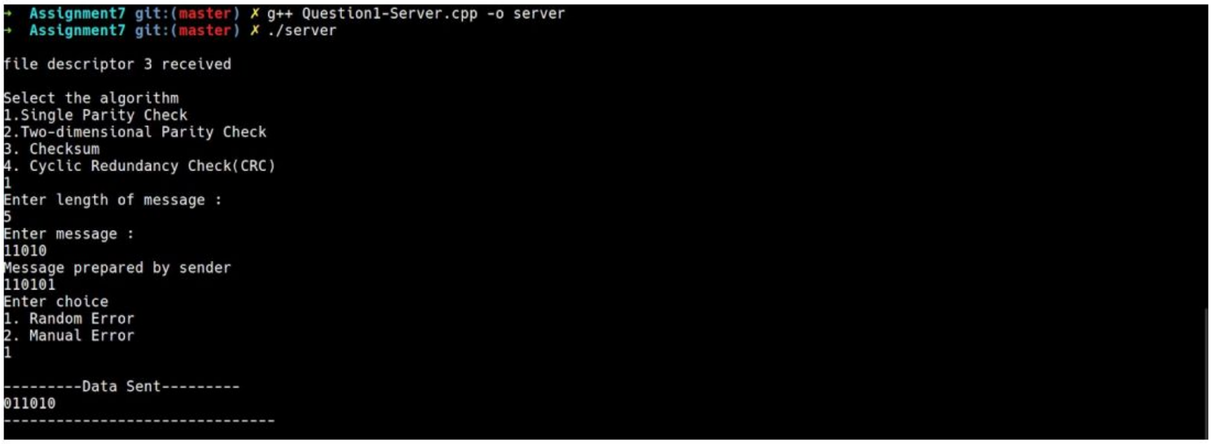
```
}

// Driver code
int main()
{
    string file;
    cout<<"Enter file name :"<<endl;
    cin>>file;

    ifstream t(file);
    string str((istreambuf_iterator<char>(t)),
istreambuf_iterator<char>());

    cout<<str<<endl;
    cout<<"Choose the coding algorithm to use :\n1. Huffman
Coding\n2. Shannon Fano Coding"<<endl;

    int choice;
    cin>>choice;

    if(choice==1){
        string encodedString, decodedString;
        calcFreq(str, str.length());
        HuffmanCodes(str.length());
        cout << "Character With there Frequencies:\n";
        for (auto v=codes.begin(); v!=codes.end(); v++)
            cout << v->first <<' ' << v->second << endl;

        for (auto i: str)
            encodedString+=codes[i];

        cout << "\nEncoded Huffman data:\n" << encodedString << endl;

        decodedString = decode_file(minHeap.top(), encodedString);
        cout << "\nDecoded Huffman Data:\n" << decodedString << endl;

    }
    else if(choice==2){
        map<char, int> syms;
        int len = str.length();
        int i;
        char cur;

        for(i=0;i<len;i++){
            cur = str[i];
            if(syms.find(cur)!=syms.end()){
                syms[cur] = syms[cur]+1;
            }
            else{
                syms[cur]=1;
            }
        }

        p = new node[syms.size()];

        int n, j;
        float total = 0;
        string ch;
        node temp;

        n = syms.size();
        // Input number of symbols
        cout << "Total number of symbols\t: ";
        cout << n << endl;

        float *x  = new float[n];
        i=0;

        map <char, int> symIndex;


        cout<<"Occurences of vaious symbols : "<<endl;
        for (map<char,int>::iterator it = syms.begin(); it !=
syms.end(); ++it) {
            p[i].sym += it->first;
            p[i].pro = it->second/(float)len;
            cout<<p[i].sym<<" "<<it->second<<endl;
            symIndex[it->first] = i;
            i++;
        }
        // Input symbols

        // Sorting the symbols based on
        // their probability or frequency
        sortByProbability(n, p);

        for (i = 0; i < n; i++)
            p[i].top = -1;

        // Find the shannon code
        shannon(0, n - 1, p);

        // Display the codes
        display(n, p);

        cout<<"Encoded file : "<<endl;
        int index;
        string encoding="";
        for(i=0;i<len;i++){
            string c = string(1, str[i]);

            cout<<shannonRev[c];
            encoding = encoding+shannonRev[c];
        }

        cout<<endl;
        cout<<"Length of encoded string = "<<encoding.length()<<endl;

        cout<<endl;
        cout<<"Decoded file : "<<endl;
        string code="";
        for(i=0;i<encoding.length();i++){
            code = code+encoding[i];
            if(shannonCodes.find(code)!=shannonCodes.end()){
                cout<<shannonCodes[code];
                code="";
            }
        }
        cout<<endl;

    }
    else{
        cout<<"Invalid choice"<<endl;
    }

    return 0;
}
```

-

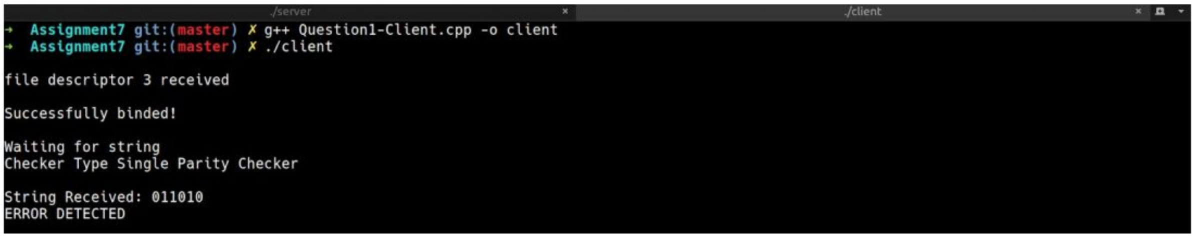
SCREENSHOTS:

QUESTION 1:

- ### Single Parity Check

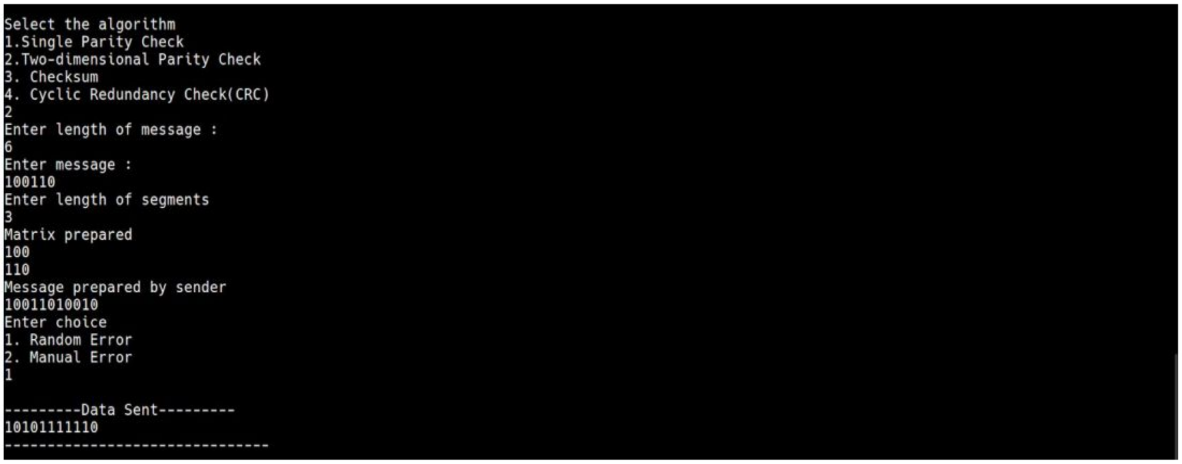


```
➜  Assignment7 git:(master) ✗ g++ Question1-Server.cpp -o server
➜  Assignment7 git:(master) ✗ ./server

file descriptor 3 received

Select the algorithm
1.Single Parity Check
2.Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check(CRC)
1
Enter length of message :
5
Enter message :
11010
Message prepared by sender
110101
Enter choice
1. Random Error
2. Manual Error
1

---------Data Sent---------
011010
-------------------------------
```

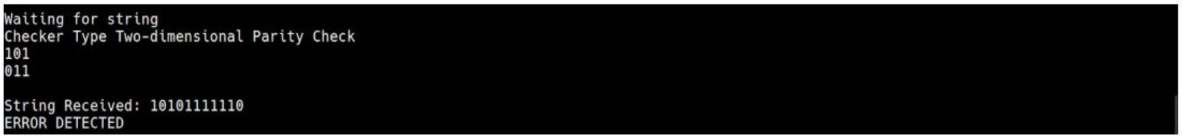


```
➜  Assignment7 git:(master) ✗ g++ Question1-Client.cpp -o client
➜  Assignment7 git:(master) ✗ ./client

file descriptor 3 received

Successfully binded!

Waiting for string
Checker Type Single Parity Checker

String Received: 011010
ERROR DETECTED
```

- ### 2D Parity Check

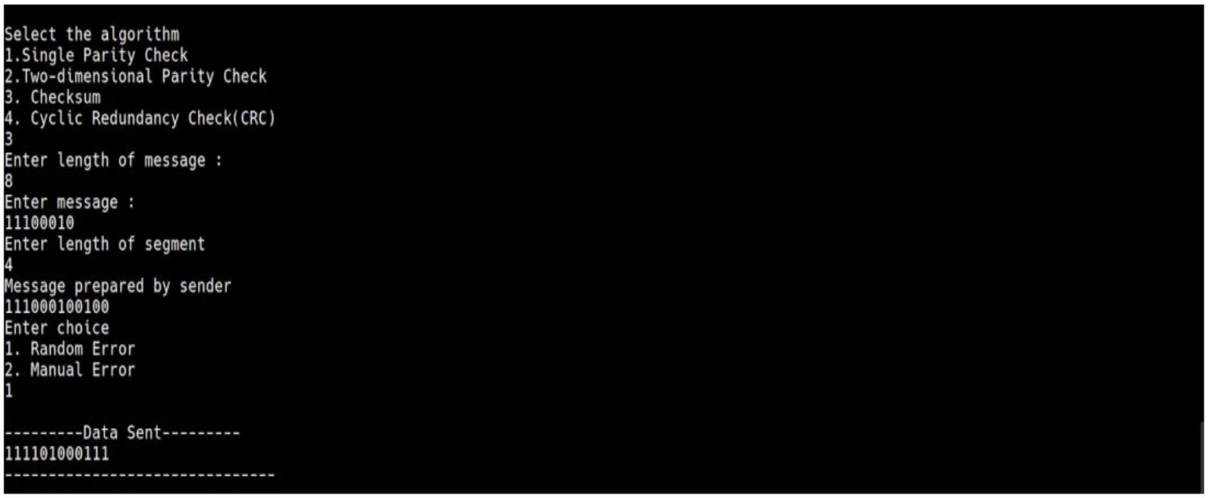


```
Select the algorithm
1.Single Parity Check
2.Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check(CRC)
2
Enter length of message :
6
Enter message :
100110
Enter length of segments
3
Matrix prepared
100
110
Message prepared by sender
10011010010
Enter choice
1. Random Error
2. Manual Error
1

---------Data Sent---------
10101111110
-------------------------------
```

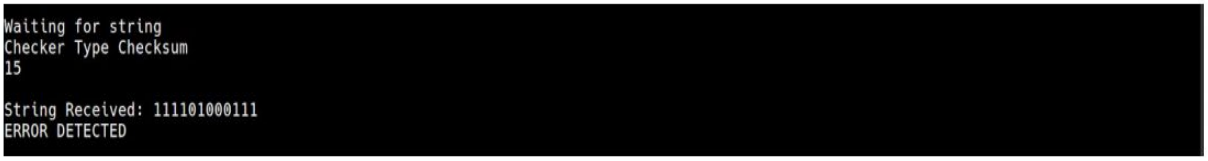


```
Waiting for string
Checker Type Two-dimensional Parity Check
101
011

String Received: 10101111110
ERROR DETECTED
```

- **Checksum**

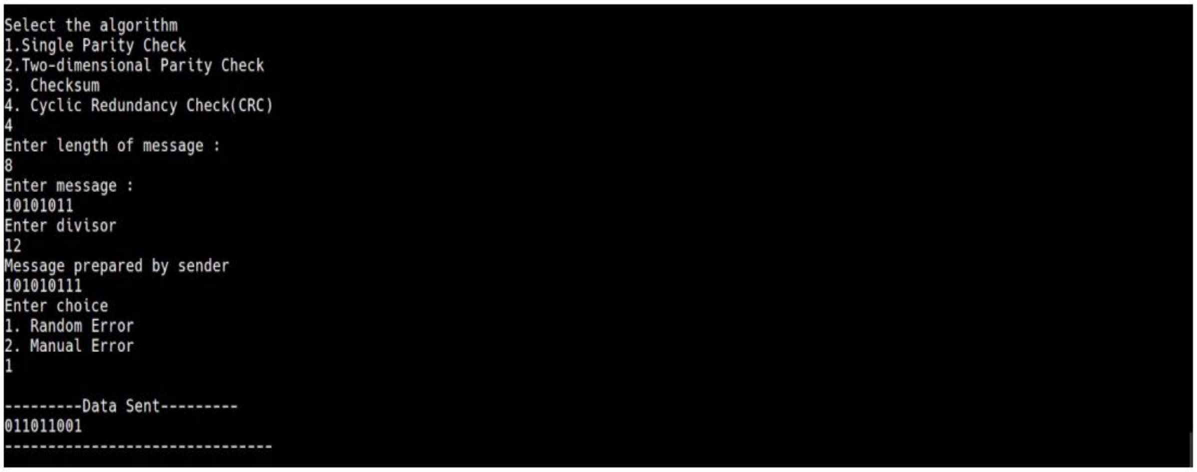


```
Select the algorithm
1.Single Parity Check
2.Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check(CRC)
3
Enter length of message :
8
Enter message :
11100010
Enter length of segment
4
Message prepared by sender
111000100100
Enter choice
1. Random Error
2. Manual Error
1

---------Data Sent---------
111101000111
-------------------------------
```

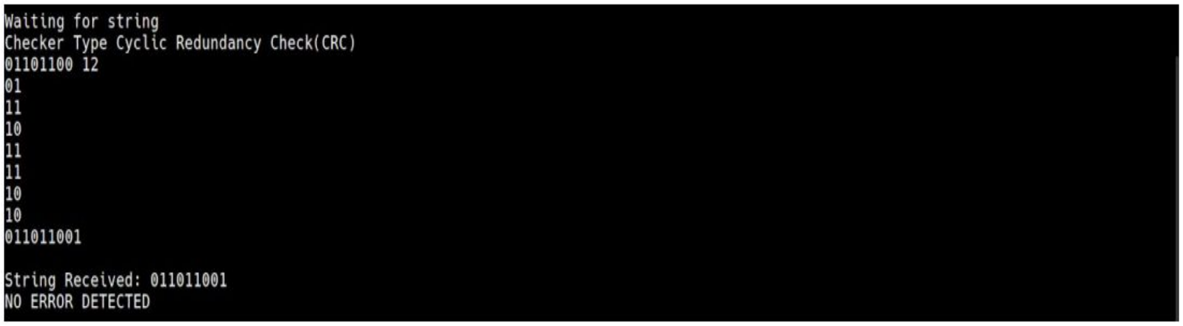


```
Waiting for string
Checker Type Checksum
15

String Received: 111101000111
ERROR DETECTED
```

- **CRC**



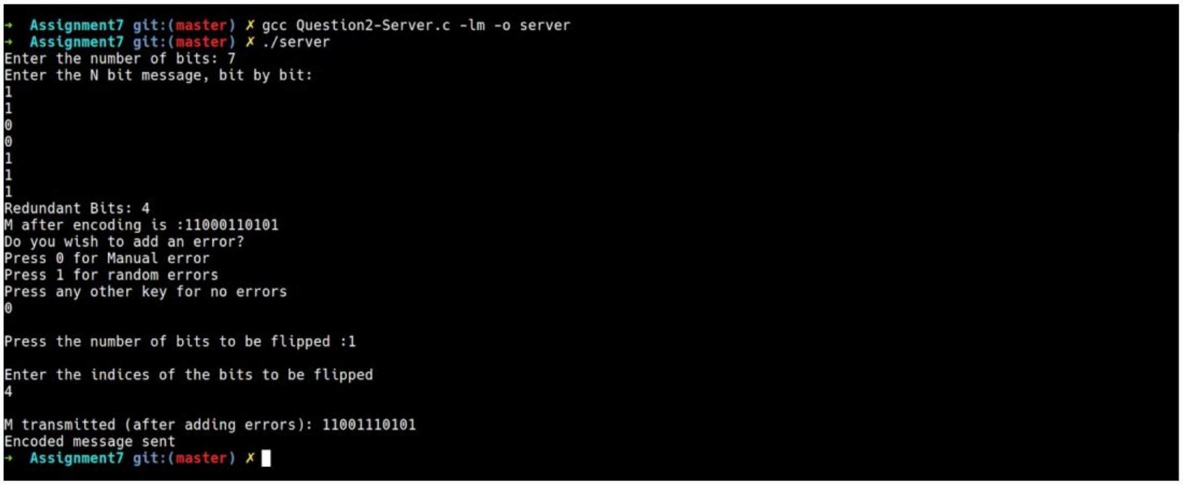
```
Select the algorithm
1.Single Parity Check
2.Two-dimensional Parity Check
3. Checksum
4. Cyclic Redundancy Check(CRC)
4
Enter length of message :
8
Enter message :
10101011
Enter divisor
12
Message prepared by sender
101010111
Enter choice
1. Random Error
2. Manual Error
1

---------Data Sent---------
011011001
-------------------------------
```



```
Waiting for string
Checker Type Cyclic Redundancy Check(CRC)
01101100 12
01
11
10
11
11
10
10
011011001

String Received: 011011001
NO ERROR DETECTED
```

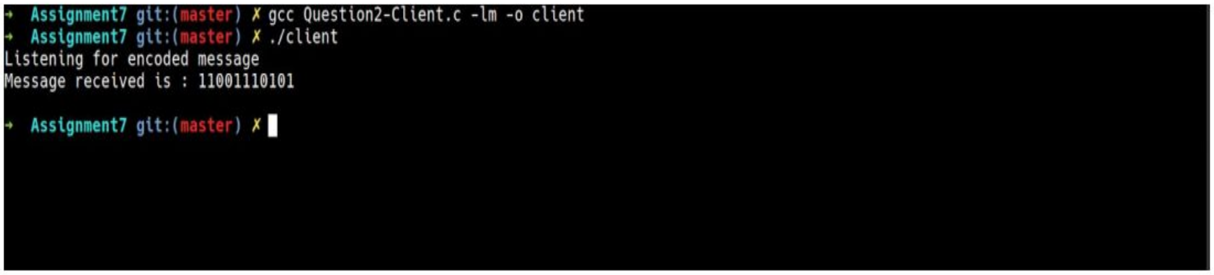
QUESTION 2

- **Server sending the encoded message**



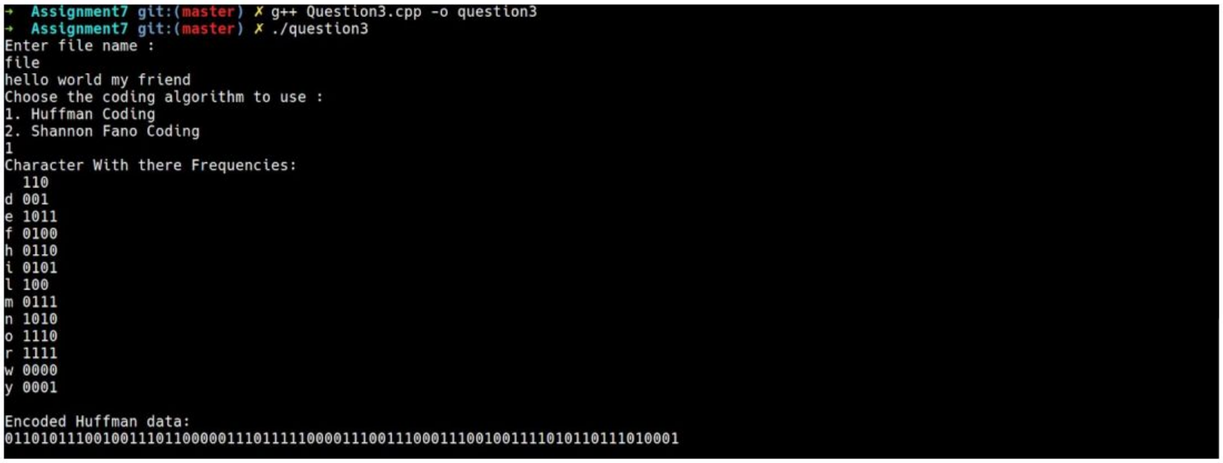
```
➜  Assignment7 git:(master) ✗ gcc Question2-Server.c -lm -o server
➜  Assignment7 git:(master) ✗ ./server
Enter the number of bits: 7
Enter the N bit message, bit by bit:
1
1
0
0
1
1
1
Redundant Bits: 4
M after encoding is :11000110101
Do you wish to add an error?
Press 0 for Manual error
Press 1 for random errors
Press any other key for no errors
0

Press the number of bits to be flipped :1

Enter the indices of the bits to be flipped
4

M transmitted (after adding errors): 11001110101
Encoded message sent
➜  Assignment7 git:(master) ✗ 
```

- **Receiving the encoded message**

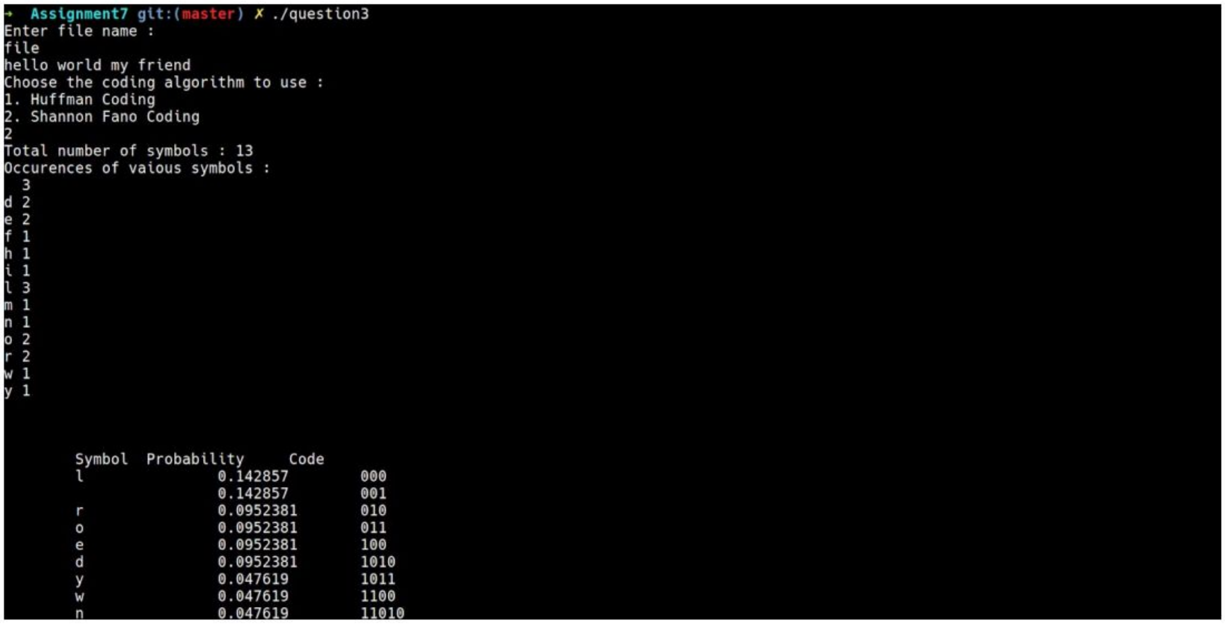


```
➜  Assignment7 git:(master) ✗ gcc Question2-Client.c -lm -o client
➜  Assignment7 git:(master) ✗ ./client
Listening for encoded message
Message received is : 11001110101

➜  Assignment7 git:(master) ✗ 
```

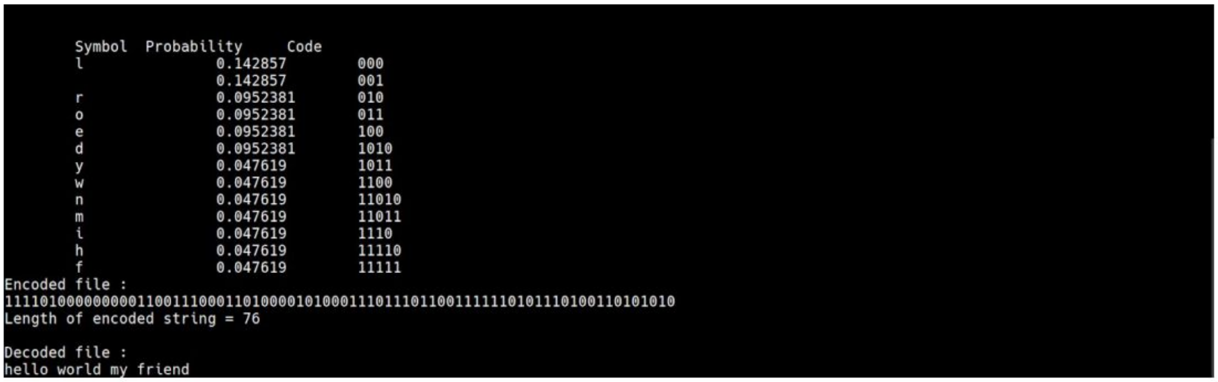
QUESTION 3



```
→  Assignment7 git:(master) ✗ g++ Question3.cpp -o question3
→  Assignment7 git:(master) ✗ ./question3
Enter file name :
file
hello world my friend
Choose the coding algorithm to use :
1. Huffman Coding
2. Shannon Fano Coding
1
Character With there Frequencies:
  110
d 001
e 1011
f 0100
h 0110
i 0101
l 100
m 0111
n 1010
o 1110
r 1111
w 0000
y 0001

Encoded Huffman data:
0110101110010011101100000111011111000011100111000111001001111010110111010001
```



```
→  Assignment7 git:(master) ✗ ./question3
Enter file name :
file
hello world my friend
Choose the coding algorithm to use :
1. Huffman Coding
2. Shannon Fano Coding
2
Total number of symbols : 13
Occurences of vaious symbols :
  3
d 2
e 2
f 1
h 1
i 1
l 3
m 1
n 1
o 2
r 2
w 1
y 1



        Symbol  Probability     Code
        l               0.142857        000
                        0.142857        001
        r               0.0952381       010
        o               0.0952381       011
        e               0.0952381       100
        d               0.0952381       1010
        y               0.047619        1011
        w               0.047619        1100
        n               0.047619        11010
```



```

        Symbol  Probability     Code
        l               0.142857        000
                        0.142857        001
        r               0.0952381       010
        o               0.0952381       011
        e               0.0952381       100
        d               0.0952381       1010
        y               0.047619        1011
        w               0.047619        1100
        n               0.047619        11010
        m               0.047619        11011
        i               0.047619        1110
        h               0.047619        11110
        f               0.047619        11111
Encoded file :
1111010000000001100111000110100001010001110111011001111110101110100110101010
Length of encoded string = 76

Decoded file :
hello world my friend
```