PROGRAMMING ASSIGNMENT 2

Program 1:

Aim : Parity bit check

Code:

// error is introduced randomly at a random location

```
/* author T.S. Vishwak */
#include<bits/stdc++.h>
using namespace std;

int main()
{
     char choice='y';
     cout<<"**********************EVEN PARITY
CHECKER**********************\n";
    do
    {
        //lets assume 8bit data is transferred and 9th bit is parity bit.
    string sender_data;
    cout<<"\nenter sender 8 bits: ";
    cin>>sender_data;

    //calculate even parity bit
    int parity=0;
    for(int i=0; i<8; i++)
    parity^=sender_data[i];

    sender_data.append("0");
    sender_data[8]=parity+'0';
    cout<<"codeword: "<<sender_data<<"\n";

    // lets select a random location and try to change it randomly to 1 or 0
    srand(time(NULL));
    int loc=rand()%8;
    int changed_bit=rand()%2;

    string receiver_data;
    receiver_data.assign(sender_data);
    receiver_data[loc]=changed_bit+'0';
    //calc parity for receiver_data
    parity=0;
    for(int i=0; i<9; i++)
```

```
    parity^=receiver_data[i];
    //changing parity from char to integer
    parity-='0';
    cout<<"receiver_data: "<<receiver_data<<"\n";

    // now let's check if sender and receiver data are the same
    if(parity==0)
    {
        cout<<"Result: data received successfully\n";
    }
    else
    cout<<"Result: data corrupted\n";
    cout<<"do you want to continue? ( y/n): ";
    cin>>choice;
    }while(choice=='y'||choice=='Y');
}
```

Output:

```
E:\sem 6\cn-tuts\parity.exe
**********EVEN PARITY CHECKER**********

enter sender 8 bits: 10010010
codeword: 100100101
receiver_data: 100100101
Result: data received successfully
do you want to continue? ( y/n): y

enter sender 8 bits: 10010010
codeword: 100100101
receiver_data: 100100101
Result: data received successfully
do you want to continue? ( y/n): y

enter sender 8 bits: 10010010
codeword: 100100101
receiver_data: 000100101
Result: data corrupted
do you want to continue? ( y/n): y

enter sender 8 bits: 10011101
codeword: 100111011
receiver_data: 100111001
Result: data corrupted
do you want to continue? ( y/n): n

--------------------------------
Process exited after 154.9 seconds with return value 0
Press any key to continue . . .
```

Conclusion: parity bit check method was implemented successfully.

Program 2:
Aim : Hamming code

Code:

```
/* author : T.S. Vishwak */
#include<bits/stdc++.h>
using namespace std;

int main()
{
    string code_word,sender_data;
    int i,m=4;
    char choice='y';
    do
    {

cout<<"*********************************************************************
********\n";
        code_word.assign(7,0);
        cout<<"enter sender_data (4bits): ";
        cin>>sender_data;
        cout<<"processing data.....\n";

        //push sender_data elements into code_word array
        code_word[2]=sender_data[0];
        code_word[4]=sender_data[1];
        code_word[5]=sender_data[2];
        code_word[6]=sender_data[3];
        //calc even parity for redundant bits
        code_word[0]=code_word[2]^code_word[4]^code_word[6];
        code_word[1]=code_word[2]^code_word[5]^code_word[6];
        code_word[3]=code_word[4]^code_word[5]^code_word[6];

        cout<<"codeword sent: "<<code_word<<endl;
        string received_data;
        received_data.assign(code_word);
        //lets try changing any random bit to 0 or 1;
        srand(time(NULL));
        int loc=rand()%7;
        int changed_bit=rand()%2;
        received_data[loc]=changed_bit+'0';
```

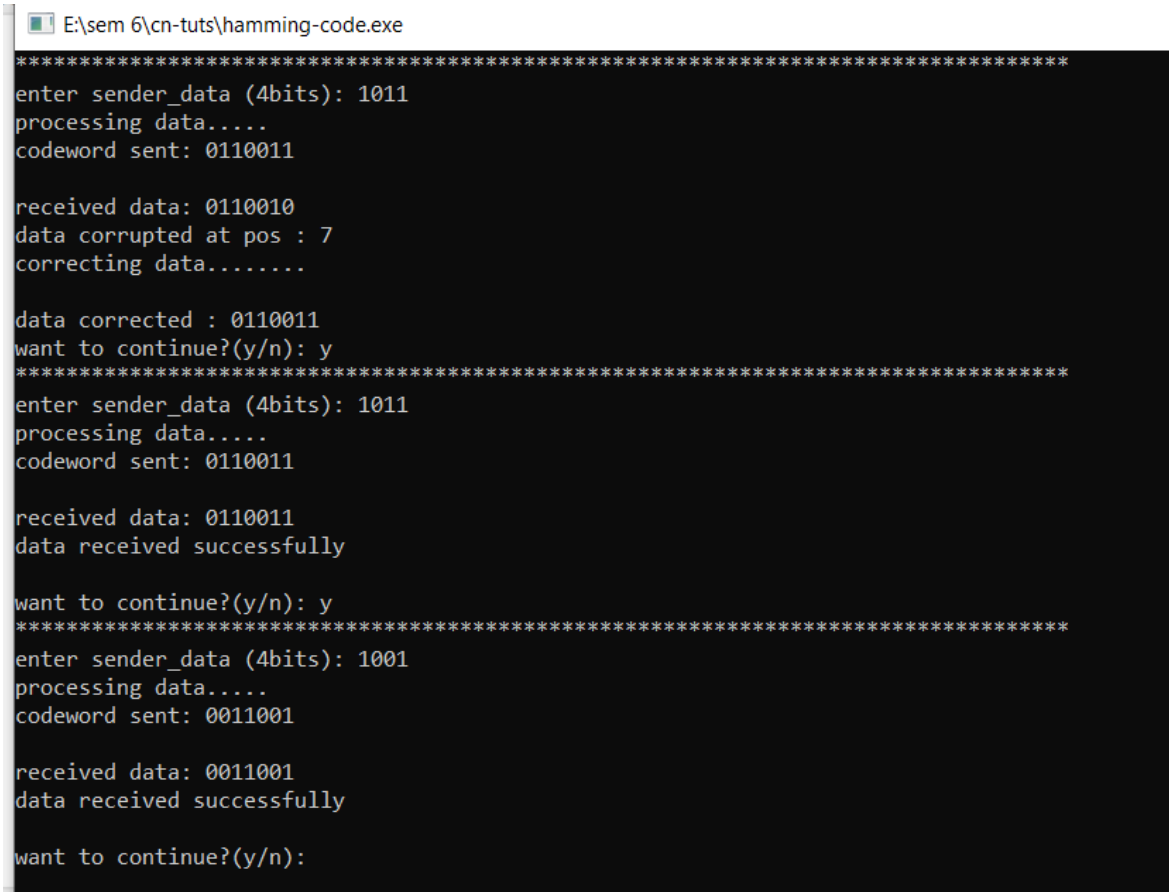
```
        cout<<"\nreceived data: "<<received_data<<endl;
        //calculate redundant bits at receiver side now;
        int c[3];
        c[0]=received_data[0]^received_data[2]^received_data[4]^received_data[6];
        c[1]=received_data[1]^received_data[2]^received_data[5]^received_data[6];
        c[2]=received_data[3]^received_data[4]^received_data[5]^received_data[6];
        //cout<<c[2]<<c[1]<<c[0]<<endl;

        //converting c to an integer
        int pos=0;
         for(i=0;i<3;i++)
         {
             pos+=pow(2,i)*c[i];
         }
        if(pos==0)
        {
            cout<<"data received successfully\n";
        }
        else
        {
            //data is corrupted at pos c2c1c0

            cout<<"data corrupted at pos : "<<pos;
            cout<<"\ncorrecting data........\n\n";
            pos--;// bcoz of 0-indexing
            //complementing it
            if(received_data[pos]=='1')
            received_data[pos]='0';
            else
            received_data[pos]='1';

            cout<<"data corrected : "<<received_data;
        }
        cout<<"\nwant to continue?(y/n): ";
        cin>>choice;
    }while(choice=='y'||choice=='Y');
}
```

Output:



Conclusion : hamming code was implemented successfully.

Program 3

Aim: Cyclic Redundancy Code (CRC)

Code :

```
//https://technicalpickout.com/cpp-code-to-implement-cyclic-redundancy-check/
//minor changes made
#include <bits/stdc++.h>
using namespace std;

void division(int *temp,int *g,int &fs,int &gs)
{
    int i,j,k;
    for (i = 0; i < fs; i++)
    {
        j = 0; k = i;
        if (temp[k] >= g[j])
        {
            for (j = 0, k = i; j < gs; j++, k++)
            {
                if ((temp[k] == 1 && g[j] == 1) || (temp[k] == 0 && g[j] == 0))
                    temp[k] = 0;
                else
                    temp[k] = 1;
            }
        }
    }
}
int main()
{
    int i, j, k, l;

    // Generator
    int gs;
    cout << "\n Enter the length of Generator(Divisor): ";
    cin >> gs;
    int g[gs];
    cout << "\n Enter the Generator(Divisor):";
    for (i = 0; i < gs; i++)
    {
        cin >> g[i];
    }
```

```
// Frame
int fs;
cout << "\n Enter length of the data: ";
cin >> fs;
int f[fs+gs];
cout << "\n Enter the data:";
for (i = 0; i < fs; i++)
{
    cin >> f[i];
}
cout << "\n Sender Side:";
//Append 0's
int rs = gs - 1;
cout << "\n Number of 0's to be appended: " << rs;
for (i = fs; i < fs + rs; i++)
{
    f[i] = 0;
}
int temp[fs+gs];
 for (i = 0; i < fs+gs; i++)
 {
     temp[i] = f[i];
 }
cout << "\n Message after appending 0's :";
for (i = 0; i < fs + rs; i++)
{
    cout << temp[i];
}
//Division
division(temp,g,fs,gs);

//CRC
int crc[rs];
for (i = 0, j = fs; i < rs; i++, j++)
{
    crc[i] = temp[j];
}
cout << "\n CRC bits: ";
for (i = 0; i < rs; i++)
{
    cout << crc[i];
}
cout << "\n Transmitted Data: ";
```

```
int tf[fs+rs];
 for (i = 0; i < fs; i++)
 {
     tf[i] = f[i];
 }
for (i = fs, j = 0; i < fs + rs; i++, j++)
{
    tf[i] = crc[j];
}
for (i = 0; i < fs + rs; i++)
{
    cout << tf[i];
}
cout << "\n Receiver side : ";
cout << "\n Received Data: ";
for (i = 0; i < fs + rs; i++)
{
    cout << tf[i];
}
 for (i = 0; i < fs + rs; i++)
 {
     temp[i] = tf[i];
 }

//Division
division(temp,g,fs,gs);

cout << "\n Remainder: ";
int rrem[fs+rs];
for (i = fs, j = 0; i < fs + rs; i++, j++)
{
    rrem[j] = temp[i];
}
for (i = 0; i < rs; i++)
{
    cout << rrem[i];
}
int flag = 0;
for (i = 0; i < rs; i++)
{
    if (rrem[i] != 0)
    {
        flag = 1;
```

```
        }
    }
    if (flag == 0)
    {
        cout << "\n Since Remainder is 0, hence the"
                " Message Transmitted from Sender to Receiver is Correct";
    }
    else
    {
        cout << "\n Since Remainder is not 0,"
        " hence the Message Transmitted from Sender to Receiver contains Error";
    }
}
```

Output:

```
E:\sem 6\cn-tuts\crc.exe

Enter the length of Generator(Divisor): 4

Enter the Generator(Divisor):1 1 0 0

Enter length of the data: 8

Enter the data:1 1 0 1 0 0 0 0

Sender Side:
Number of 0's to be appended: 3
Message after appending 0's :11010000000
CRC bits: 100
Transmitted Data: 11010000100
Receiver side :
Received Data: 11010000100
Remainder: 000
Since Remainder is 0, hence the Message Transmitted from Sender to Receiver is Correct
--------------------------------
Process exited after 33.46 seconds with return value 0
Press any key to continue . . .
```

Conclusion : CRC was implemented successfully.