

**Course Code: BCSE308P**

**Course Name: Computer Networks Lab**

**In Lab Assessment – 3**

**Name: Shivam Dave**

**Reg. No.: 21BCB0107**

**Problem Statement:**

**To implement error detection (CRC) and error correction techniques(Hamming code).**

**Code:**

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

void hamming(int sender[],int receiver[],int n)//hamming code

{

    int paritybitS1 = sender[0] xor sender[1] xor sender[3] xor sender[4] xor sender[6];

    int paritybitS2 = sender[0] xor sender[2] xor sender[3] xor sender[5] xor sender[6];

    int paritybitS3 = sender[1] xor sender[2] xor sender[3] xor sender[7];

    int paritybitS4 = sender[4] xor sender[5] xor sender[6] xor sender[7];

    int paritybitR1 = receiver[0] xor receiver[1] xor receiver[3] xor receiver[4] xor receiver[6];

    int paritybitR2 = receiver[0] xor receiver[2] xor receiver[3] xor receiver[5] xor receiver[6];

    int paritybitR3 = receiver[1] xor receiver[2] xor receiver[3] xor receiver[7];

    int paritybitR4 = receiver[4] xor receiver[5] xor receiver[6] xor receiver[7];

    int syndrome1 = paritybitR1 xor paritybitS1;

    int syndrome2 = paritybitR2 xor paritybitS2;

    int syndrome3 = paritybitR3 xor paritybitS3;

    int syndrome4 = paritybitR4 xor paritybitS4;

    int Syndrome = syndrome4\*(8) + syndrome3\*(4) + syndrome2\*(2) + syndrome1\*(1);

    cout<<"Syndrome is: "<<Syndrome<<endl;

    int pos = -1;

    int parityPos = -1;

    switch (Syndrome){

    case 1:{

        pos = -2;

        parityPos = 1;

        break;

    }

    case 2:{

        pos = -2;

        parityPos = 2;

        break;

    }

    case 3:{

        pos = 1;

        // parityPos = 1;

        break;

    }

    case 4:{

        pos = -1;

        parityPos = 3;

        break;

    }

    case 5:{

        pos = 2;

        // parityPos = 1;

        break;

    }

    case 6:{

        pos = 3;

        // parityPos = 2;

        break;

    }

    case 7:{

        pos = 4;

        // parityPos = 1;

        break;

    }

    case 8:{

        pos = -1;

        parityPos = 4;

        break;

    }

    case 9:{

        pos = 5;

        // parityPos = 1;

        break;

    }

    case 10:{

        pos = 6;

        // parityPos = 2;

        break;

    }

    case 11:{

        pos = 7;

        // parityPos = 1;

        break;

    }

    case 12:{

        pos = 8;

        // parityPos = 2;

        break;

    }

    default:{

        pos = -1;

        parityPos = -1;

        break;

    }

    }

    if(pos == -1 && parityPos == -1){

        cout<<"No Error"<<endl;

    }

    if(parityPos != -1){

        cout<<"Parity position is wrong at position: "<<parityPos<<endl;

    }else{

        cout<<endl;

        cout<<"The sender bits were:   ";

        for(int i = n-1; i>=0; i--){

            cout<<sender[i];

        }

        cout<<endl;

        cout<<"The receiver bits were: ";

        for(int i = n-1; i>=0; i--){

            cout<<receiver[i];

        }

        cout<<endl;

        cout<<"Data bit wrong position: "<<pos<<endl;

        // cout<<endl;

        if(receiver[pos-1] == 1){

            receiver[pos-1] = 0;

        }else{

            receiver[pos-1] = 1;

        }

        cout<<"The corrected receiver bits are: ";

        for(int i = n-1; i>=0; i--){

            cout<<receiver[i];

        }

        cout<<endl;

    }

}

//xor for CRC

void XOR(int\* total, int total\_bits, int \* generator, int generate\_n, int index){

    for(int i = 0; i<generate\_n; i++){

        if(total[index+i] == generator[i]){

            total[index+i] = 0;

        }else{

            total[index+i] = 1;

        }

    }

}

//function for CRC

void CRC(int\* total, int total\_bits, int \* generator, int generate\_n){

    for(int i = 0; i<=total\_bits - generate\_n; i++){

        if(total[i] == 1){

            XOR(total, total\_bits,generator,generate\_n, i);

            i--;

            // cout<<"After XOR: ";

            // for(int i = 0; i<total\_bits; i++){

            //     cout<<total[i];

            // }

            // cout<<endl;

        }

    }

}

void c\_r\_c(int sender[], int receiver[],int n)

{

     int n\_generator;

    cout<<"Enter the number of bits in generator polynomial: ";

    cin>>n\_generator;

    int generator[n\_generator];

    cout<<"Enter the generator polynomial: ";

    for(int i = 0; i<n\_generator;i++){

        cin>>generator[i];

    }

    cout<<endl;

    cout<<"The sender bits:   ";

    for(int i = 0; i<n; i++){

        cout<<sender[i];

    }

    cout<<endl;

    cout<<"The receiver bits: ";

    for(int i = 0; i<n; i++){

        cout<<receiver[i];

    }

    cout<<endl;

    int total\_bits = n + n\_generator -1;

    int total\_sender[total\_bits];

    for(int i = 0; i<n; i++){

        total\_sender[i] = sender[i];

    }

    for(int i = n; i<total\_bits; i++){

        total\_sender[i] = 0;

    }

    // cout<<"After appending zeros to sender bits: ";

    // for(int i = 0; i<total\_bits; i++){

    //     cout<<total\_sender[i];

    // }

    // cout<<endl;

    CRC(total\_sender, total\_bits, generator, n\_generator);

    // cout<<"After performing CRC on sender: ";

    // for(int i = 0; i<total\_bits; i++){

    //     cout<<total\_sender[i];

    // }

    // cout<<endl;

    cout<<"Check word for sender is: ";

    for(int i = 0; i<n\_generator-1; i++){

        cout<<total\_sender[n+i];

    }

    cout<<endl;

    cout<<"The sender bits are:   ";

    for(int i = 0; i<n; i++){

        cout<<sender[i];

    }

    for(int i = 0; i<n\_generator-1; i++){

        cout<<total\_sender[n+i];

    }

    cout<<endl;

    int total\_receiver[total\_bits];

    for(int i = 0; i<n; i++){

        total\_receiver[i] = receiver[i];

    }

    for(int i = 0; i<n\_generator; i++){

        total\_receiver[n+i] = total\_sender[n+i];

    }

    cout<<"The receiver bits are: ";

    for(int i = 0; i<total\_bits; i++){

        cout<<total\_receiver[i];

    }

    cout<<endl;

    CRC(total\_receiver, total\_bits, generator, n\_generator);

    cout<<"After performing CRC on receiver: ";

    for(int i = 0; i<total\_bits; i++){

        cout<<total\_receiver[i];

    }

    cout<<endl;

    // cout<<"Check word for receiver is: ";

    bool found\_error = false;

    for(int i = 0; i<n\_generator; i++){

        // cout<<total\_receiver[n+i-1];

        if(total\_receiver[n+i-1]){

            // cout<<total\_receiver[n+i-1]<<" ";

            // cout<<"Index: "<<n+i<<endl;

            found\_error = true;

            break;

        }

    }

    if(found\_error){

        cout<<"Error in transmission"<<endl;

    }else{

        cout<<"No error in transmission"<<endl;

    }

}

int main()

{

    int n;

    cout<<"Enter the number of bits: ";

    cin>>n;

    cout<<"Enter the sender bits: ";

    int sender[n];

    for(int i = n-1; i>=0; i--){

        cin>>sender[i];

        if(sender[i]>1 )

        {

            cout<<"You have to enter binary digits only!";

            break;

        }

    }

    cout<<"Enter the receiver bits: ";

    int receiver[n];

    for(int i = n-1; i>=0; i--){

        cin>>receiver[i];

        // if(receiver[i]!=1 || receiver[i]!=0)

        // {

        //     cout<<"You have to enter binary digits only!";

        // }

    }

    int choose;

    cout<<"Press 1 if you want to go for Hamming code error correction or Press 0 for checksum:";

    cin>>choose;

    switch(choose)

    {

        case 1:{

            hamming(sender,receiver,n);

        }

        case 0:{

            c\_r\_c(sender,receiver,n);

        }

        default:

        {

            cout<<"You didn't press 1 or 0";

            break;

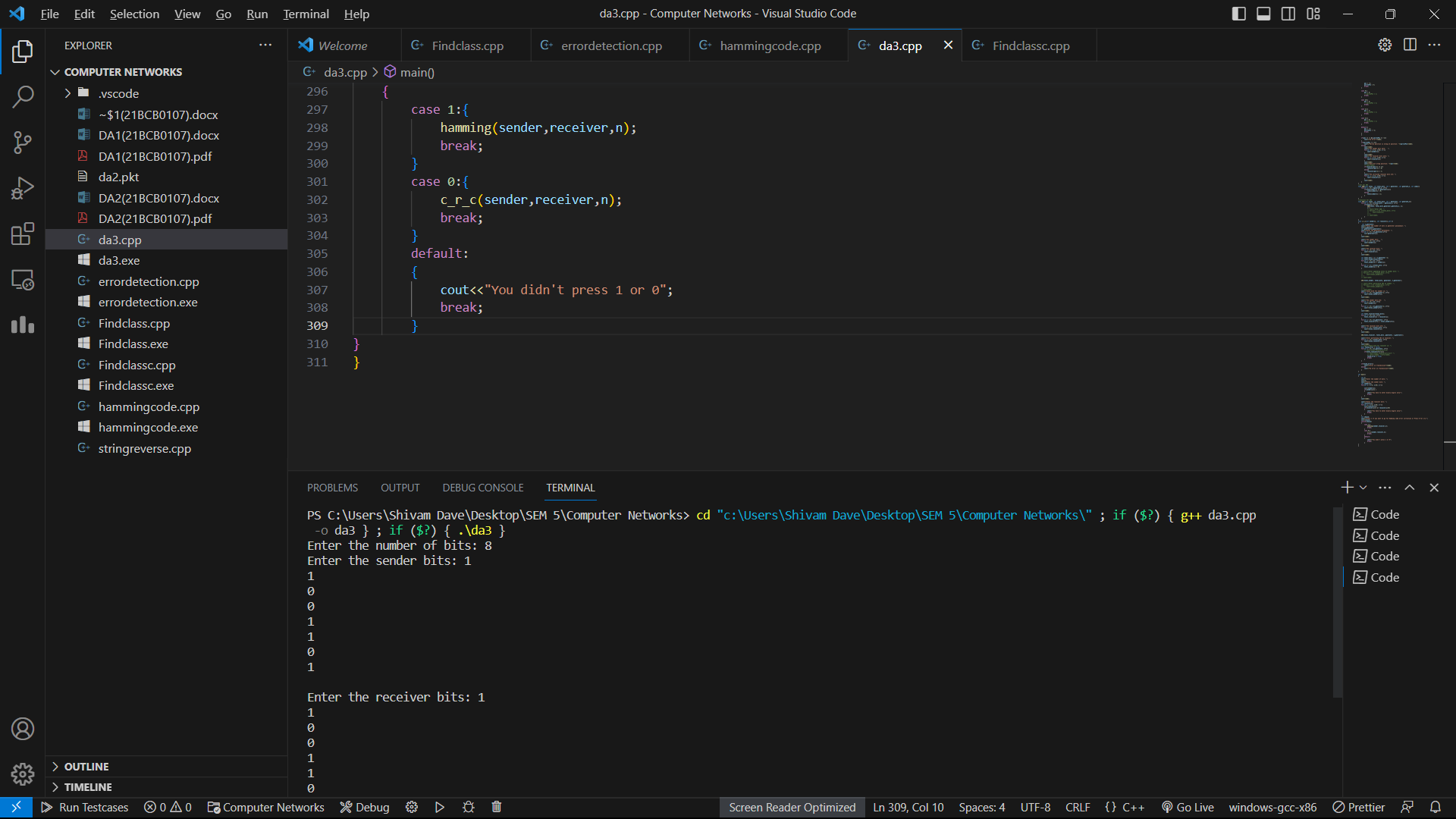
        }

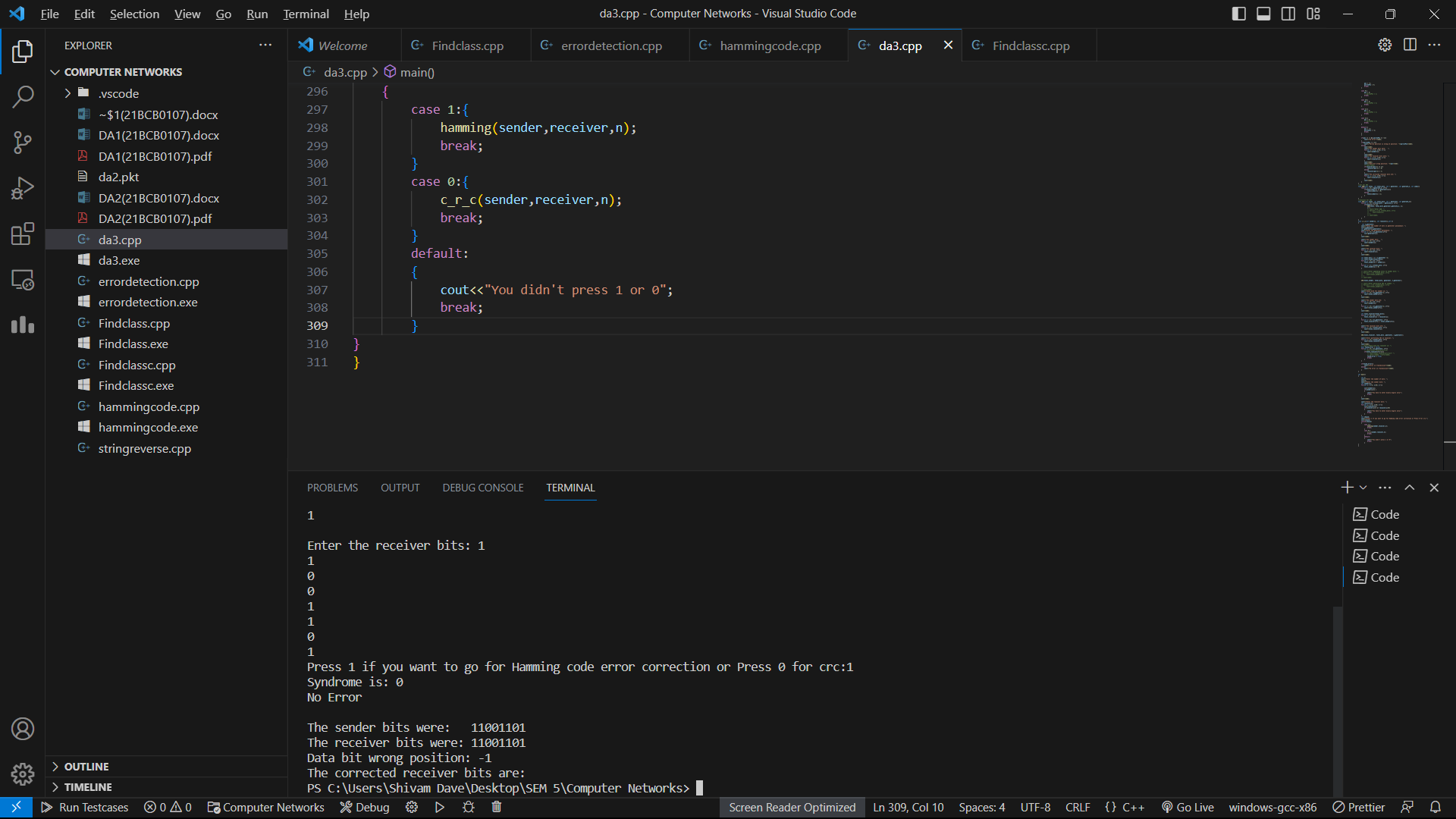
}

}

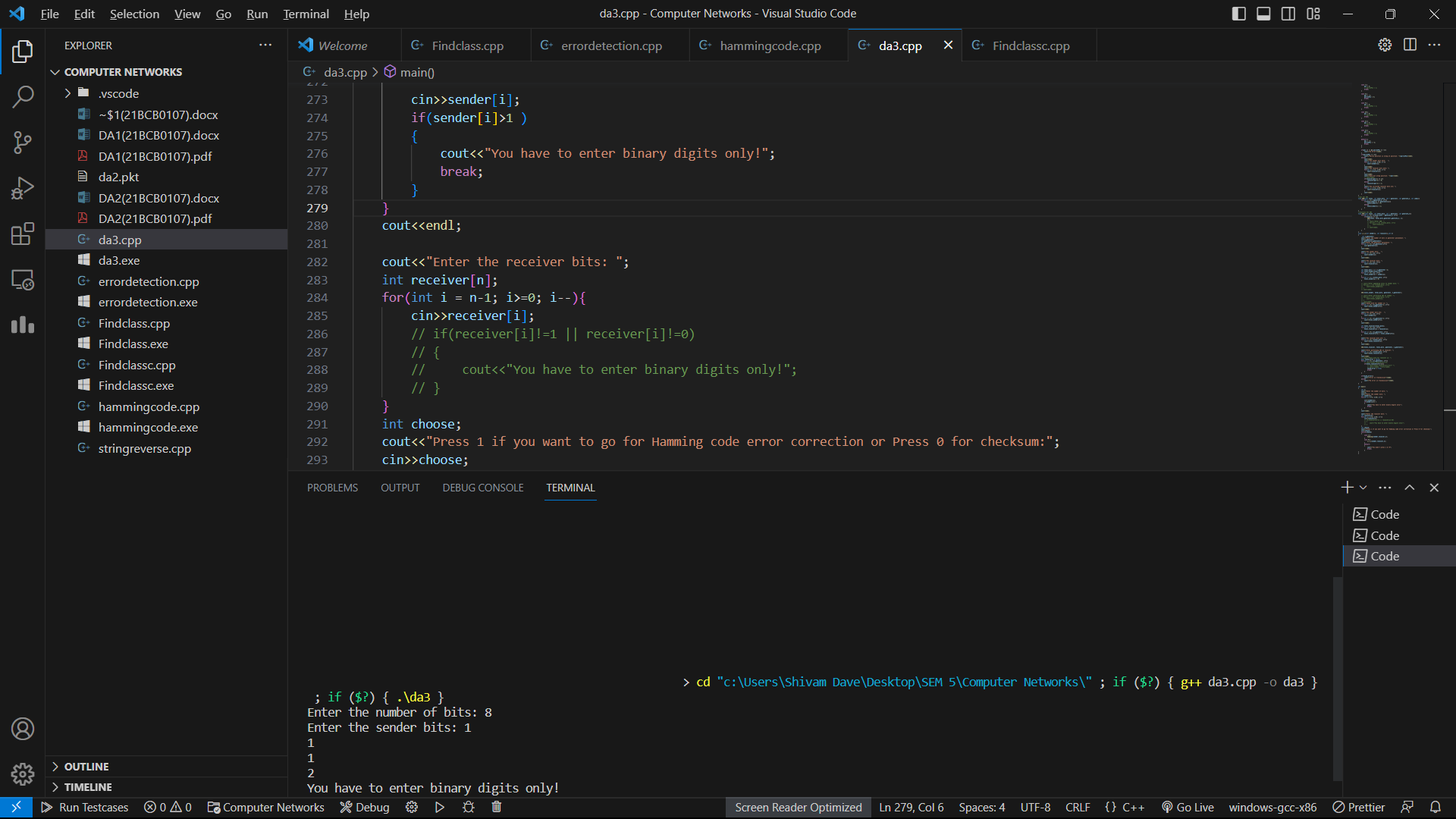
**Output(s):**

**For hamming code where receiver input is same as sender input:**

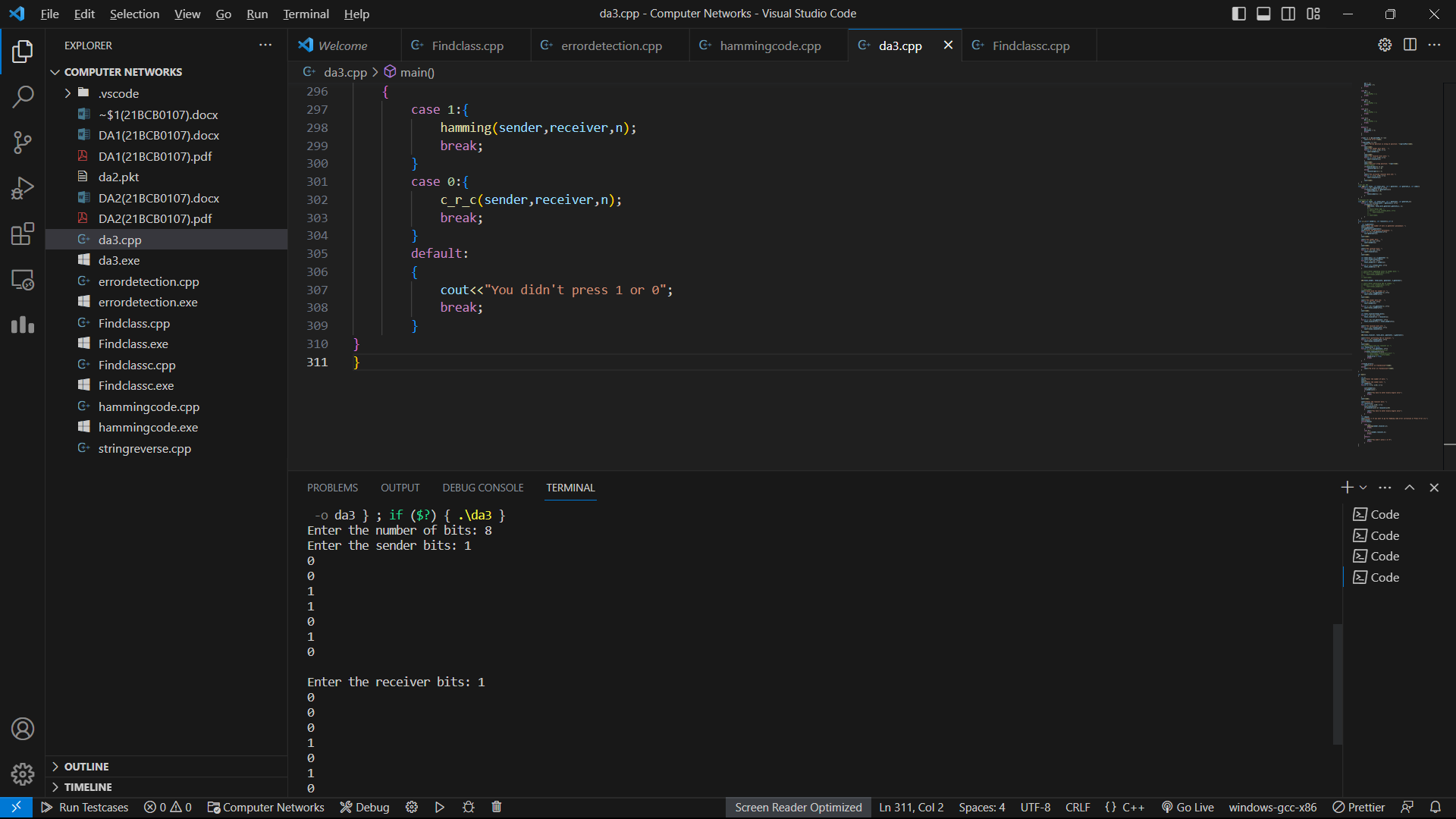


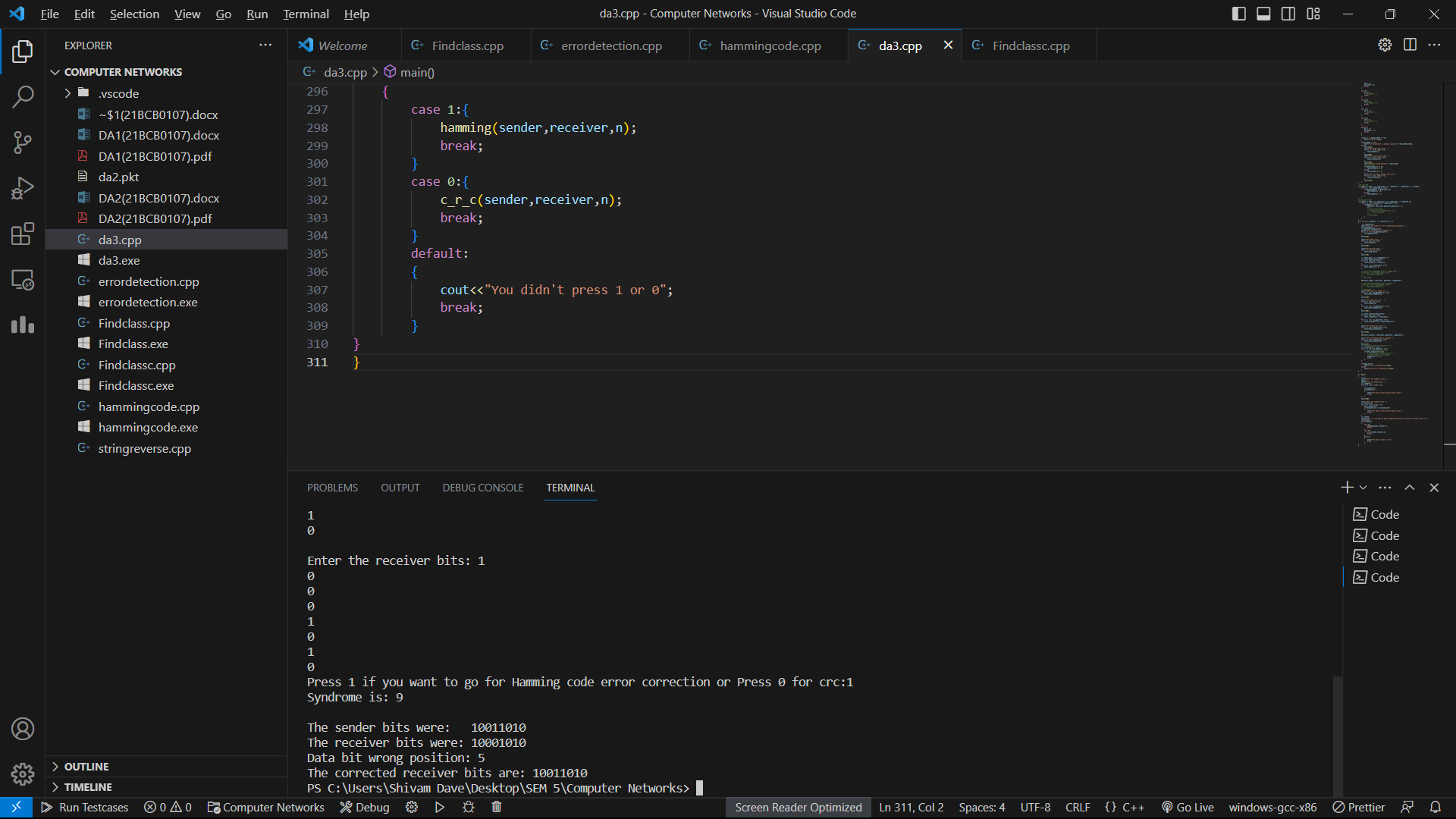


**For inputting a number that is not Binary digits:**



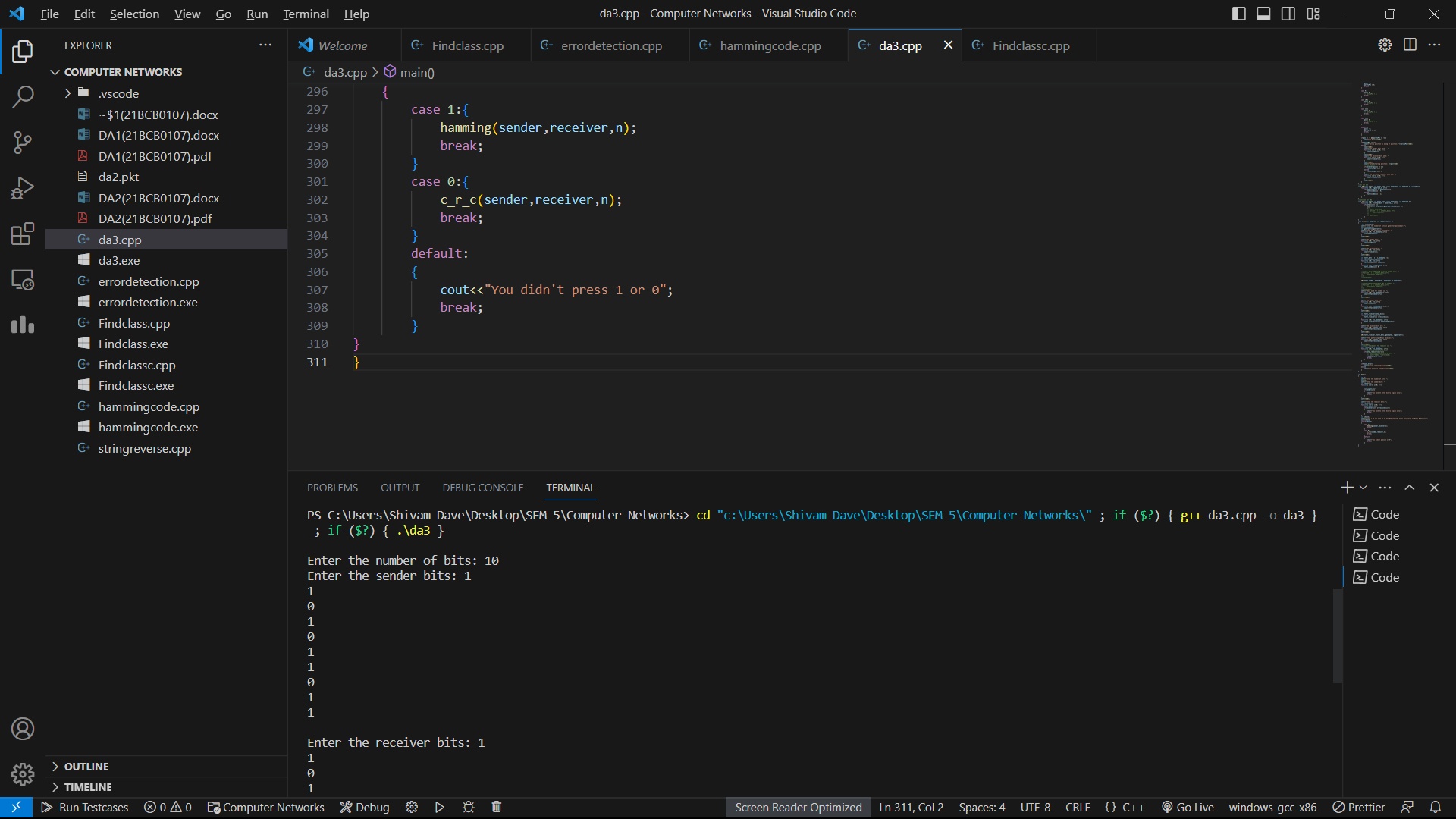
**For incorrect input in receiver’s side in Hamming code:**

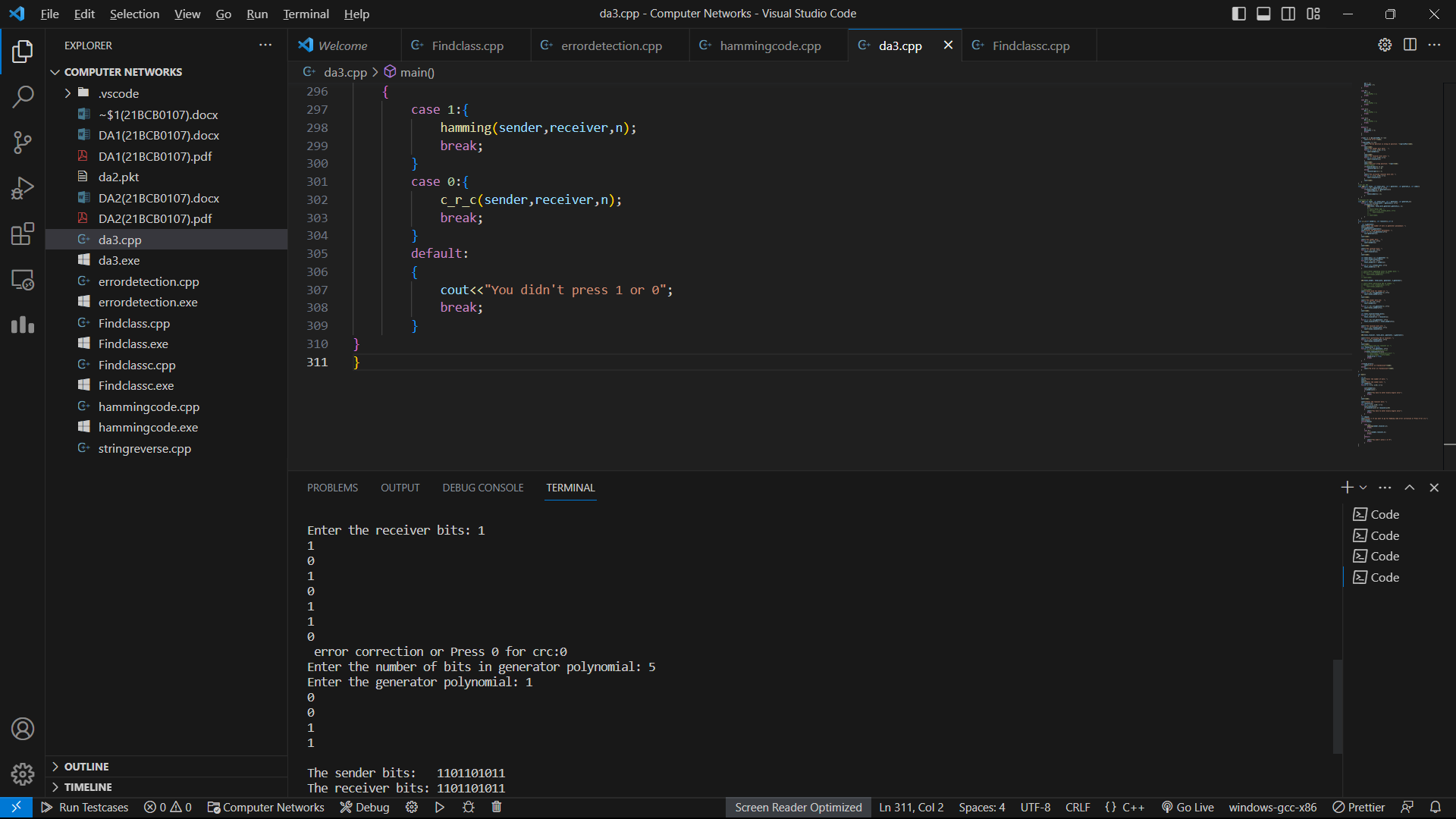


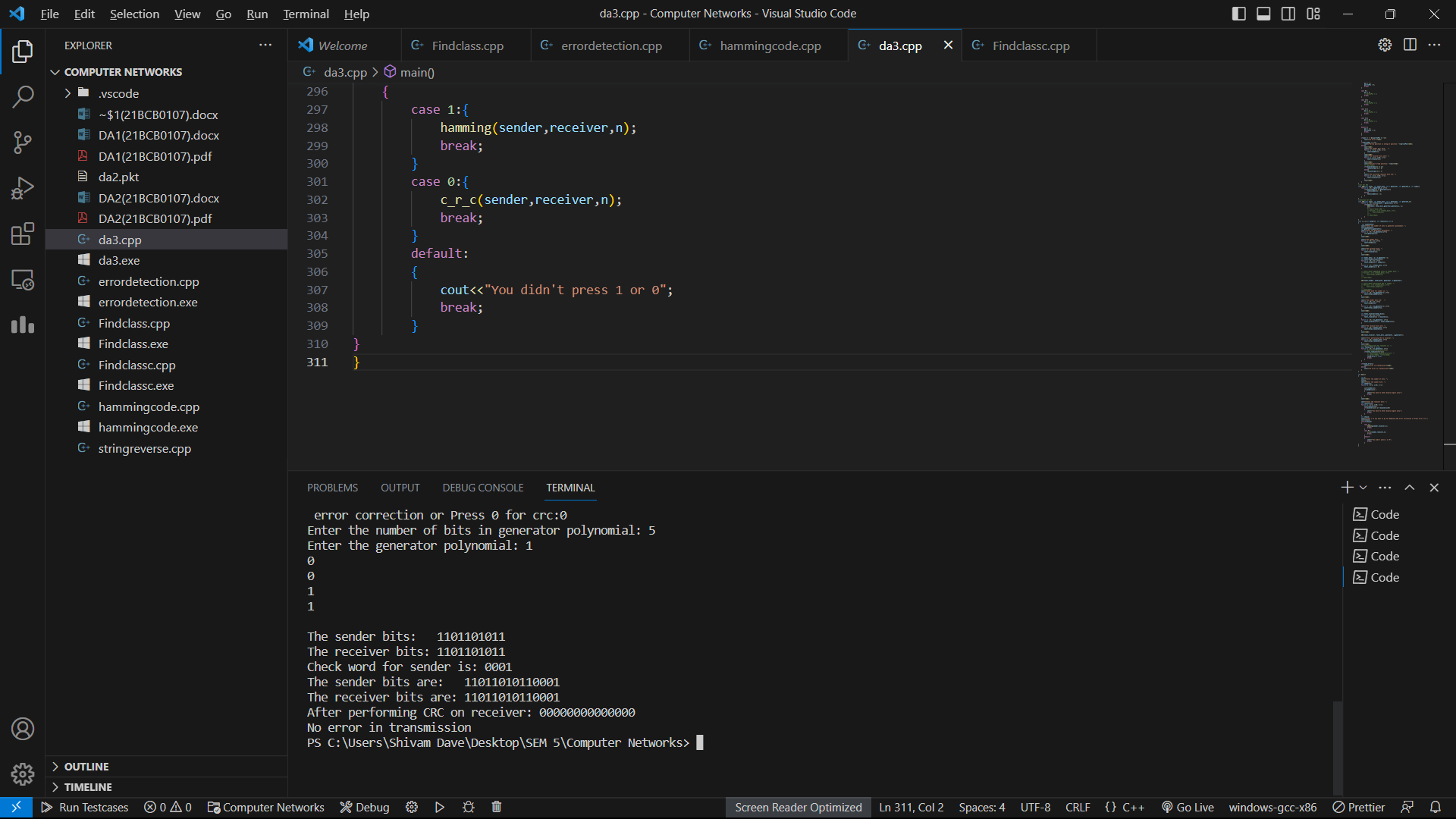


**For CRC:**

**Correct receiver side’s input:**







**For incorrect receiver side’s input:**

