**DELHI TECHNOLOGIAL UNIVERSITY**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



**CS208: COMPUTER NETWORKS**

**LAB FILE**

SUBMITTED TO:

MS. GULL KAUR,

ASSISTANT PROFESSOR,

CSE DEPARTMENT

SUBMITTED BY:

KSHITIJ SINHA,

23/CS/225,

A4, G1

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Experiment Title** | **Page Number** | **Sign** |
| 1 | Bitstuffing, bytestuffing, and character stuffing | **4** |  |
| 2 | CRC Implementation | **6** |  |
|  |  |  |  |

**Experiment 1**

**AIM:** Implement Framing techniques – BitStuffing, ByteStuffing, Character Stuffing

**THEORY:**

Bit Stuffing : Bit stuffing is a method used in data communication to avoid confusion between data and special control signals (like start or end markers). When a specific sequence of bits appears in the data an extra bit is added to break the pattern. This prevents the receiver from mistaking the data for control information. Once the data is received the extra bit is removed restoring the original message. This technique helps ensure accurate data transmission.

Byte Stuffing : Byte stuffing is the same as bit stuffing the only difference is that instead of a single bit, one byte of data is added to the message to avoid confusion between data and special control signals. This ensures accurate message transmission without misinterpreting the data

**CODE:**

#include <bits/stdc++.h>

using namespace std;

string bitstuff(string &s){

    string header="01111110";

    string tail="01111110";

    string scpy=header;

    scpy.append(s);

    scpy.append(tail);

    s=scpy;

    int n=s.length();

    for(int i=8; i<n-8; i++){

        if(s[i]=='1'&&s[i-1]=='1'&&s[i-2]=='1'&&s[i-3]=='1'&&s[i-4]=='1'&&s[i-5]=='1'&&s[i-6]=='1'&&s[i-7]=='1'){

            s.insert(i+1,1, '0');

            n++;

        }

    }

    return s;

}

// string bit\_destuff(string& s){

//     string scp;

//     for(int i=8; i<s.length()-8; i++){

//         char ss=s[i];

//         if(s[i]=='1'&&s[i-1]=='1'&&s[i-2]=='1'&&s[i-3]=='1'&&s[i-4]=='1'&&s[i-5]=='1'&&s[i-6]=='1'&&s[i-7]=='1'){

//             scp.push\_back(ss);

//             i++;

//         }

//         else{

//             scp.push\_back(ss);

//         }

//     }

//     s=scp;

//     return s;

// }

string bytestuff(string& s){

    string flag="0110";

    string escape="1101";

    //F->EF, E->EE

    string sc=s;

    for(int i=0; i<sc.length(); i++){

        if(sc[i]=='0'&&sc[i+1]=='1'&&sc[i+2]=='1'&&sc[i+3]=='0'){

            sc.insert(i, 1, '1');

            sc.insert(i+1, 1, '1');

            sc.insert(i+2, 1, '0');

            sc.insert(i+3, 1, '0');

            i+=4;

        }

        else if(sc[i]=='1'&&sc[i+1]=='1'&&sc[i+2]=='0'&&sc[i+3]=='1'){

            sc.insert(i, 1, '1');

            sc.insert(i+1, 1, '1');

            sc.insert(i+2, 1, '0');

            sc.insert(i+3, 1, '0');

            i+=4;

        }

    }

    string ssss=flag;

    ssss.append(sc);

    ssss.append(flag);

    s=ssss;

    return s;

}

int main(){

    string s="10001";

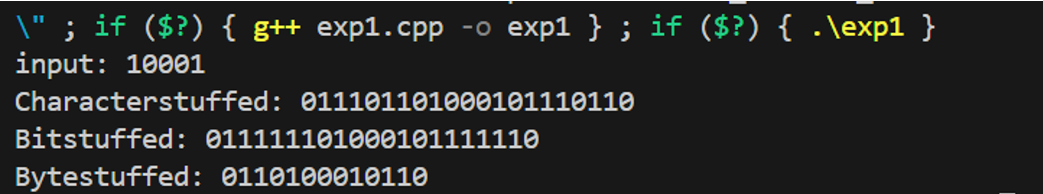
    cout<<"Bitstuffed: "<<bitstuff(s)<<endl;

     s="10001";

    cout<<"Bytestuffed: "<<bytestuff(s);

}

**OUTPUT:**

****

**Experiment 2**

**AIM:** Implement Cyclic Redundancy Check (CRC)

**THEORY:**

CRC or Cyclic Redundancy Check is a method of detecting accidental changes/errors in the communication channel.

Cyclic Redundancy Check (CRC) is an error-detection technique used in computer networks and communication systems to ensure data integrity during transmission. It detects accidental changes to raw data by adding a short, fixed-length binary sequence (the CRC code) to the data being transmitted.

Total bits sent = number of pure bits + number of redundant bits appended (equal to the degree of polynomial)

**CODE:**

#include <bits/stdc++.h>

using namespace std;

int main(){

    string a,b;

    cout<<"Enter Dataword"<<endl;

    cin>>a;

    cout<<"Enter Polynomial in binary encoding"<<endl;

    cin>>b;

    string ac=a;

    int appendd=b.length()-1;

    for(int i=0; i<appendd; i++){

        ac.push\_back('0');

    }

    int aa=0, bb=0;

    int exp=0;

    for(int i=ac.length()-1; i>-1; i--){

        if(ac[i]=='1'){

            aa+=pow(2,exp);

        }

        exp++;

    }

    exp=0;

    for(int i=b.length()-1; i>-1; i--){

        if(b[i]=='1'){

            bb+=pow(2, exp);

        }

        exp++;

    }

    int rem=aa%bb;

    for(int i=ac.length()-1; i>=ac.length()-appendd; i--){

        if(rem%2==0){

            ac[i]='0';

        }

        else{

            ac[i]='1';

        }

        rem/=2;

    }

    cout<<"String received?"<<endl;

    string c;

    cin>>c;

    if(ac==ac){

        cout<<"Correct"<<endl;

    }

    else{

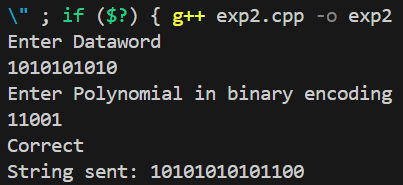
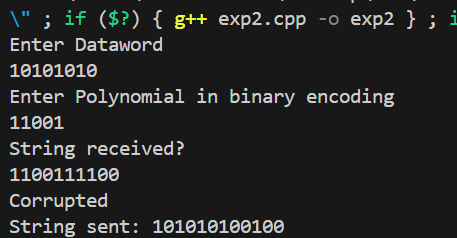
        cout<<"Corrupted"<<endl;

    }

    cout<<"String sent: "<<ac<<endl;

}

**OUTPUT**

****

**Experiment 3**

**AIM:** Implement Error Correction and Error Detection using Hamming code

**THEORY:**

Hamming bits are used in computer networks for error detection and correction. They are part of a mechanism known as Hamming Code, which helps identify and correct single-bit errors in transmitted data.

**CODE:**

#include <bits/stdc++.h>

using namespace std;

string even\_check(string s){

    string p;

    int cnt1=0, cnt2=0, cnt4=0, cnt8=0;

    for(int i=0; i<8; i+=2){

        if(s[i]=='1') cnt1++;

    }

    if(s[1]=='1') cnt2++;

    if(s[2]=='1') cnt2++;

    if(s[5]=='1') cnt2++;

    if(s[6]=='1') cnt2++;

    for(int i=3; i<=6; i++){

        if(s[i]=='1') cnt4++;

    }

    if(s[7]=='1') cnt8++;

    cnt1%=2;

    cnt2%=2;

    cnt4%=2;

    cnt8%=2;

    char out[12];

    if(cnt1==1){

        out[0]='1';

    }

    else{

        out[0]='0';

    }

    if(cnt2==1){

        out[1]='1';

    }

    else{

        out[1]='0';

    }

    if(cnt4==1){

        out[3]='1';

    }

    else{

        out[3]='0';

    }

    if(cnt8==1){

        out[7]='1';

    }

    else{

        out[7]='0';

    }

    int idx=0;

    for(int i=0; i<12; i++){

        if(i==0||i==1||i==3||i==7){

            continue;

        }

        else{

            out[i]=s[idx];

            idx++;

        }

    }

    string ans;

    for(int i=0; i<12; i++){

        ans.push\_back(out[i]);

    }

    return ans;

}

string odd\_check(string s){

    string p;

    int cnt1=0, cnt2=0, cnt4=0, cnt8=0;

    for(int i=0; i<8; i+=2){

        if(s[i]=='1') cnt1++;

    }

    if(s[1]=='1') cnt2++;

    if(s[2]=='1') cnt2++;

    if(s[5]=='1') cnt2++;

    if(s[6]=='1') cnt2++;

    for(int i=3; i<=6; i++){

        if(s[i]=='1') cnt4++;

    }

    if(s[7]=='1') cnt8++;

    cnt1%=2;

    cnt2%=2;

    cnt4%=2;

    cnt8%=2;

    char out[12];

    if(cnt1==1){

        out[0]='0';

    }

    else{

        out[0]='1';

    }

    if(cnt2==1){

        out[1]='0';

    }

    else{

        out[1]='1';

    }

    if(cnt4==1){

        out[3]='0';

    }

    else{

        out[3]='1';

    }

    if(cnt8==1){

        out[7]='0';

    }

    else{

        out[7]='1';

    }

    int idx=0;

    for(int i=0; i<12; i++){

        if(i==0||i==1||i==3||i==7){

            continue;

        }

        else{

            out[i]=s[idx];

            idx++;

        }

    }

    string ans;

    for(int i=0; i<12; i++){

        ans.push\_back(out[i]);

    }

    return ans;

}

int main(){

    string s;

    cout<<"Enter a string of length 8: "<<endl;

    cin>>s;

    cout<<"Enter even\_check received: "<<endl;

    string p;

    cin>>p;

    cout<<"Enter odd\_check received: "<<endl;

    string q;

    cin>>q;

    string pp=even\_check(s);

    string qq=odd\_check(s);

    if(pp==p){

        cout<<"Correct even\_check received"<<endl;

    }

    else{

        cout<<"Incorrect even check received"<<endl;

    }

    if(qq==q){

        cout<<"Correct odd\_check received"<<endl;

    }

    else{

        cout<<"Incorrect odd check received"<<endl;

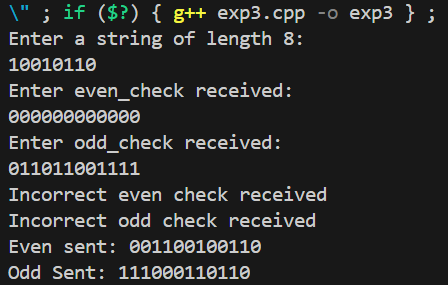
    }

    cout<<"Even sent: "<<even\_check(s)<<endl;

    cout<<"Odd Sent: "<<odd\_check(s)<<endl;

}

**OUTPUT:**

****