# CODE :-

```
/*
 * Problem Statement :-
 *  Write a program for error detection and correction for 7/8 bits
ASCII codes using
 *  Hamming Codes or CRC.
 */

#include <bits/stdc++.h>
using namespace std;

class CRC
{
    private:
        string data, key;

    public:
        void input();
        string xor1(string, string);
        string mod2div(string, string);
        string encodedData();
        string remainder(string);
};

void CRC::input()
{
    cout<<"\n\t\t Enter Data (string) : ";
    cin>>data;

    cout<<"\n\t\t Enter Key (string) : ";
    cin>>key;
}

string CRC::xor1(string a, string b)
{
    string result = "";
    int n = b.length();

    for(int i = 1; i < n; i++)
    {
        if (a[i] == b[i])
            result += "0";
        else
            result += "1";
    }
    return result;
}

string CRC::mod2div(string dividend, string divisor)
{
    int pick = divisor.length();
```

```
    string tmp = dividend.substr(0, pick);

    int n = dividend.length();

    while (pick < n)
    {
        if (tmp[0] == '1')
            tmp = xor1(divisor, tmp) + dividend[pick];
        else
            tmp = xor1(string(pick, '0'), tmp) + dividend[pick];

        pick += 1;
    }

    if (tmp[0] == '1')
        tmp = xor1(divisor, tmp);
    else
        tmp = xor1(string(pick, '0'), tmp);

    return tmp;
}

string CRC::encodedData()
{
    int l_key = key.length();

    string appended_data = (data +string(l_key - 1, '0'));

    string remainder = mod2div(appended_data, key);

    string codeword = data + remainder;

    cout << "\n\t\t Remainder : "<< remainder << "\n";
    cout << "\n\t\t Encoded Data (Data + Remainder) :"<< codeword <<
 "\n";

    return codeword;
}

class HammingCode
{
    private:
        vector<int> msgBit;
        char p;
    public:
        void input();
        vector<int> generateHammingCode(int, int);
        void findHammingCode();
        void checkError(vector<int>&, int);
};

void HammingCode::input()
{
```

```
    int m;
    cout<<"\n\t\t Enter number of bits in Message : ";
    cin>>m;
    msgBit.resize(m);
    cout<<"\n\t\t Enter Message (space separated) : ";
    for(int i=0; i<m; i++)
    {
        cin>>msgBit[i];
    }
    cout<<"\n\t\t Enter Parity (e/o) : ";
    cin>>p;
}

vector<int> HammingCode::generateHammingCode(int m, int r )
{
    vector<int> hammingCode(r + m);

    for (int i = 0; i < r; ++i)
    {
        hammingCode[pow(2, i) - 1] = -1;
    }

    int j = 0;

    for (int i = 0; i < (r + m); i++)
    {
        if (hammingCode[i] != -1)
        {
            hammingCode[i] = msgBit[j];
            j++;
        }
    }

    for (int i = 0; i < (r + m); i++)
    {
        if (hammingCode[i] != -1)
            continue;

        int x = log2(i + 1);
        int one_count = 0;

        for (int pos = i + 2; pos <= (r + m); ++pos)
        {
            if (pos & (1 << x))
            {
                if (hammingCode[pos - 1] == 1)
                {
                    one_count++;
                }
            }
        }

        if (one_count % 2 == 0)
```

```
        {
            if(p == 'e')
                hammingCode[i] = 0;
            else
                hammingCode[i] = 1;
        }
        else
        {
            if(p == 'e')
                hammingCode[i] = 1;
            else
                hammingCode[i] = 0;
        }
    }

    return hammingCode;
}

void HammingCode::checkError(vector<int>& receivedCode, int r)
{
    vector<int> pos;
    vector<int> parity;
    for(int i=0; i<r; i++)
    {
        pos.push_back(pow(2,i)-1);
    }
    for (unsigned int i = 0; i < pos.size(); i++)
    {
        int x = log2(pos[i] + 1);
        int one_count = 0;

        for (unsigned int j = pos[i]; j <= receivedCode.size(); j++)
        {
            if (j & (1 << x))
            {
                if (receivedCode[j - 1] == 1)
                {
                    one_count++;
                }
            }
        }

        if (one_count % 2 == 0)
        {
            if(p == 'e')
                parity.push_back(0);
            else
                parity.push_back(1);
        }
        else
        {
            if(p == 'e')
                parity.push_back(1);
```

```
            else
                parity.push_back(0);
        }
    }
    int cnt = 0;
    for(unsigned int i=0; i<parity.size(); i++)
    {
        cnt += parity[i]*pow(2,i);
    }
    if(cnt == 0)
    {
        cout<<"\n\t\t No Error...!!"<<endl;
    }
    else
    {
        cout<<"\n\t\t Error...!! Error present at position "<<cnt<<e
ndl;

        receivedCode[cnt-1] = receivedCode[cnt-1] == 1 ? 0:1;
        cout<<"\n\t\t Corrected Received Code : ";
        for(unsigned int i=0; i<receivedCode.size(); i++)
        {
            cout<<receivedCode[i]<<" ";
        }
    }
}

void HammingCode::findHammingCode()
{
    int m = msgBit.size();

    int r = 1;

    while (pow(2, r) < (m + r + 1))
    {
        r++;
    }

    vector<int> ans = generateHammingCode(m, r);

    cout << "\n\t\t Message bits are : ";
    for (unsigned int i = 0; i < msgBit.size(); i++)
        cout << msgBit[i] << " ";

    cout << "\n\n\t\t Receiver side Hamming code is : ";
    for (unsigned int i = 0; i < ans.size(); i++)
        cout << ans[i] << " ";

    cout<<"\n\n\t\t Enter Received Code of length("<<ans.size()<<")
: ";

    vector<int> receivedData(ans.size());
    for(unsigned int i=0; i<ans.size(); i++)
```

```
    {
        cin>>receivedData[i];
    }

    checkError(receivedData, r);
}

int main()
{
    int choice;

    while(true)
    {
        cout<<"\n === Main-
Menu ===  \n\t 1. CRC  \n\t 2. Hamming Code  \n\t 3. Exit \n";
        cout<<"\n\t Enter Your Choice : ";
        cin>>choice;

        if(choice == 1)
        {
            CRC c1;
            string str;

            c1.input();

            string encodedString = c1.encodedData();

            cout<<"\n\t\t Enter received data (string): ";
            cin>>str;

            bool flag = true;
            if(str.length() != encodedString.length()) flag = false;
            else if (flag)
            {
                for(unsigned int i=0; i<str.length(); i++)
                {
                    if(str[i] != encodedString[i])
                    {
                        flag = false;
                        break;
                    }
                }
            }

            if(flag)
            {
                cout<<"\n\t\t Received Data is valid."<<endl;
            }
            else
            {
                cout<<"\n\t\t Error!! Received Data is Invalid."<<en
dl;
            }
```

```
        }
        else if(choice == 2)
        {
            HammingCode h1;
            h1.input();
            h1.findHammingCode();
        }
        else if(choice == 3)
        {
            cout<<"\n\n\t\t\t === Thank You ===";
            exit(0);
        }
        else
        {
            cout<<"\n\t Enter correct choice...!!"<<endl;
        }
    }

    return 0;
}
```

## OUTPUT :-

```
=== Main-Menu ===
   1. CRC
   2. Hamming Code
   3. Exit

   Enter Your Choice : 1

      Enter Data (string) : 100100

      Enter Key (string) : 1101

      Remainder : 001

      Encoded Data (Data + Remainder) :100100001

      Enter received data (string): 100100001

      Received Data is valid.

=== Main-Menu ===
   1. CRC
   2. Hamming Code
   3. Exit

   Enter Your Choice : 1
```

```
      Enter Data (string) : 100100

      Enter Key (string) : 1101

      Remainder : 001

      Encoded Data (Data + Remainder) :100100001

      Enter received data (string): 100000001

      Error!! Received Data is Invalid.

=== Main-Menu ===
   1. CRC
   2. Hamming Code
   3. Exit

   Enter Your Choice : 2

      Enter number of bits in Message : 7

      Enter Message (space separated) : 1 0 1 1 0 0 1

      Enter Parity (e/o) : e

      Message bits are : 1 0 1 1 0 0 1

      Receiver side Hamming code is : 1 0 1 0 0 1 1 1 0 0 1

      Enter Received Code of length(11) : 1 0 1 0 0 1 1 1 0 0 1

      No Error...!!

=== Main-Menu ===
   1. CRC
   2. Hamming Code
   3. Exit

   Enter Your Choice : 2

      Enter number of bits in Message : 7

      Enter Message (space separated) : 1 0 1 1 0 0 1

      Enter Parity (e/o) : e

      Message bits are : 1 0 1 1 0 0 1

      Receiver side Hamming code is : 1 0 1 0 0 1 1 1 0 0 1

      Enter Received Code of length(11) : 1 0 1 0 0 0 1 1 0 0 1

      Error...!! Error present at position 6
```

```
      Corrected Received Code : 1 0 1 0 0 1 1 1 0 0 1

=== Main-Menu ===
   1. CRC
   2. Hamming Code
   3. Exit

   Enter Your Choice : 3


         === Thank You ===
```