**Experiment 03: Cyclic Redundancy Check Code**

***Sender Side Code***

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

class DataStream

{

public:

int n; ///Number of bits in data

int k; ///Number of bits in g(x)

vector<int> data;

vector<int> gx;

DataStream(int n,int k,vector<int> data,vector<int> gx)

{

this->n=n;

this->k=k;

this->data=data;

this->gx=gx;

cout<<"\n\_\_\_\_Initialisation Data\_\_\_"<<endl;

cout<<"Data: ";

for(int i=0; i<n; i++) {cout<<data[i]<<" ";}

cout<<endl;

cout<<"G(X): ";

for(int i=0; i<k; i++){ cout<<gx[i]<<" ";}

cout<<endl;

}

};

class CodeWord:public DataStream

{

public:

vector<int> quot;

vector<int> curr\_divident;

CodeWord(int n,int k,vector<int> data,vector<int> gx):DataStream(n,k,data,gx)

{

vector<int> temp(data.begin(),data.begin()+k);

curr\_divident=temp;

AppendBits();

}

void AppendBits()

{

for(int i=0; i<k-1; i++)

{

data.push\_back(0);

}

cout<<"\n\_\_\_After appending "<<k-1<<" 0s to data\_\_\_"<<endl;

cout<<"Updated Data: ";

for(int i=0; i<data.size(); i++){cout<<data[i]<<" ";}

cout<<endl;

}

void FormCodeWord()

{

int j=0;

for(int i=n; i<data.size(); i++)

{data[i]=curr\_divident[j++];}

cout<<"\nRemainder after division i.e CRC is: ";

for(int i=0;i<curr\_divident.size();i++){

cout<<curr\_divident[i]<<" ";

}

cout<<"\n\n\_\_\_Data sent to receiver\_\_\_"<<endl;

cout<<"CODE WORD IS: ";

for(int i=0; i<data.size(); i++) cout<<data[i]<<" ";

cout<<endl;

}

void Divide()

{

int special\_break=0;

int nextbit=k;

while(nextbit<data.size())

{

/// => If data[0] i.e first bit is NOT 1 it's not divisible

/// => Append 0 to quototient and update the divident before xoring

if(curr\_divident[0]==0)

{

quot.push\_back(0);

///Remove first bit

curr\_divident.erase(curr\_divident.begin(),curr\_divident.begin()+1);

///Push the next bit into divident

curr\_divident.push\_back(data[nextbit++]);

///Continue xoring AS WE HAVE UPDATED the divident already

}

/// => Xor each element of divident

for(int i=0; i<k; i++)curr\_divident[i]=(curr\_divident[i]^gx[i]);

quot.push\_back(1);

///Remove first bit

curr\_divident.erase(curr\_divident.begin(),curr\_divident.begin()+1);

if(nextbit==data.size())

{

special\_break=1;

break;

}

curr\_divident.push\_back(data[nextbit++]);

}

if(!special\_break) curr\_divident.erase(curr\_divident.begin(),curr\_divident.begin()+1);

///\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_FORM THE CODE WORD\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

FormCodeWord();

}

};

int main()

{

int n,k,bit;

vector<int> data,gx;

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

cin>>n;

cout<<"Enter the number of bits in g(x): ";

cin>>k;

cout<<"Enter the data stream to be sent: ";

for(int i=0; i<n; i++)

{ cin>>bit;

data.push\_back(bit);

}

cout<<"Enter the g(x) stream: ";

for(int i=0; i<k; i++)

{ cin>>bit;

gx.push\_back(bit);

}

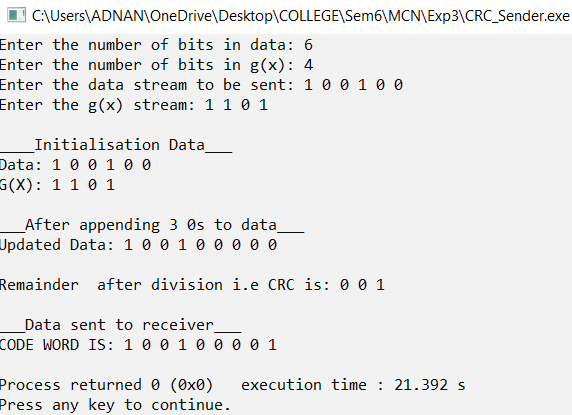
CodeWord \*c1=new CodeWord(n,k,data,gx);

c1->Divide();

return 0;

}

***Output***

******

***Receiver Side Code:***

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

class DataStream

{

public:

int n; ///Number of bits in data

int k; ///Number of bits in g(x)

vector<int> data;

vector<int> gx;

DataStream(int n,int k,vector<int> data,vector<int> gx)

{

this->n=n;

this->k=k;

this->data=data;

this->gx=gx;

cout<<"\n\_\_\_\_Data Received [Code Word] is\_\_\_"<<endl;

cout<<"Data: ";

for(int i=0; i<n; i++) {cout<<data[i]<<" ";}

cout<<endl;

cout<<"G(X): ";

for(int i=0; i<k; i++){ cout<<gx[i]<<" ";}

cout<<endl;

}

};

class CodeWord:public DataStream

{

public:

vector<int> quot;

vector<int> curr\_divident;

CodeWord(int n,int k,vector<int> data,vector<int> gx):DataStream(n,k,data,gx)

{

vector<int> temp(data.begin(),data.begin()+k);

curr\_divident=temp;

}

void Verify()

{

bool no\_error=true;

cout<<"\n\_\_\_\_\_C O N C L U S I O N\_\_\_\_\_\nRemainder obntained after division is: "<<endl;

for(int i=0;i<curr\_divident.size();i++){

cout<<curr\_divident[i]<<" ";

}

cout<<endl;

for(int i=0;i<curr\_divident.size();i++){

int bit=curr\_divident[i];

if(bit!=0){

no\_error=false;

cout<<"Remainder Obtained is NOT zero || Hence ask sender to resend the data"<<endl;

}

}

if(no\_error){

cout<<"\nData received without any ERROR"<<endl;

cout<<"\nActual Data is: ";

for(int i=0;i<data.size()-curr\_divident.size();i++){

cout<<data[i]<<" ";

}

}

}

void Divide()

{

int special\_break=0;

int nextbit=k;

/// => Iterate till no bits are left in data to be appended

while(nextbit<data.size())

{

/// => If data[0] i.e first bit is NOT 1 it's not divisible

/// => Append 0 to quototient and update the divident before xoring

if(curr\_divident[0]==0)

{

quot.push\_back(0);

///Remove first bit

curr\_divident.erase(curr\_divident.begin(),curr\_divident.begin()+1);

///Push the next bit into divident

curr\_divident.push\_back(data[nextbit++]);

///Continue xoring AS WE HAVE UPDATED the divident already

}

/// => Xor each element of divident

for(int i=0; i<k; i++)curr\_divident[i]=(curr\_divident[i]^gx[i]);

quot.push\_back(1);

///Remove first bit

curr\_divident.erase(curr\_divident.begin(),curr\_divident.begin()+1);

/// => Applied only when next bit has reach size at which there is no further bit to append

if(nextbit==data.size())

{

special\_break=1;

break;

}

///Push the next bit into divident

curr\_divident.push\_back(data[nextbit++]);

}

/// => ELIMINATE the first bit, remainig bits is remainder as we need [K-1] bits only

if(!special\_break) curr\_divident.erase(curr\_divident.begin(),curr\_divident.begin()+1);

///\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_VERIFY FROM REMAINDER\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Verify();

}

};

int main()

{

int n,k,bit;

vector<int> data,gx;

cout<<"Enter the number of bits in received code word: ";

cin>>n;

cout<<"Enter the number of bits in g(x): ";

cin>>k;

cout<<"Enter the code word received: ";

for(int i=0; i<n; i++)

{ cin>>bit;

data.push\_back(bit);

}

cout<<"Enter the g(x) stream: ";

for(int i=0; i<k; i++)

{ cin>>bit;

gx.push\_back(bit);

}

CodeWord \*c1=new CodeWord(n,k,data,gx);

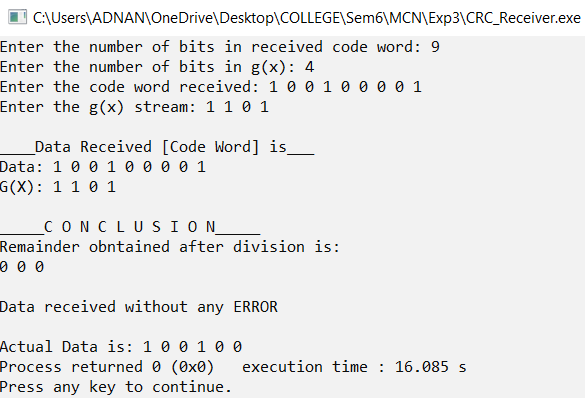
c1->Divide();

return 0;

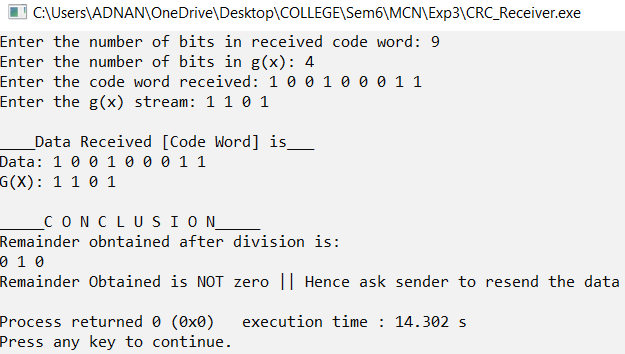
}

***Output:***

***Case 01: Without Error in Code Word***

******

***Case 01: With Error in Code Word***

******