

**Name – Parth Dipak Kulkarni**

**Programme – TY B.Tech. Computer Engineering**

**Div – A, Batch – A4, Roll No. – 80**

**PRN – 202201040007**

**Course – Computer Networks Lab**

**Assignment 2 – CRC and Hamming Code**

**Implement a program to send a massage from source to destination with error checking through CRC and Hamming code. Prepare Client and server process and show bit comparison for error checking.**

1. **CRC**

**Program –**

#include <bits/stdc++.h>

using namespace std;

// Returns XOR of 'a' and 'b' (both of the same length)

string xor1(string a, string b)

{

string result = "";

int n = b.length();

// Traverse all bits, if bits are the same, then XOR is 0, else 1

for (int i = 1; i < n; i++) {

if (a[i] == b[i])

result += "0";

else

result += "1";

}

return result;

}

// Performs Modulo-2 division

string 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(std::string(pick, '0'), tmp) + dividend[pick];

pick += 1;

}

if (tmp[0] == '1')

tmp = xor1(divisor, tmp);

else

tmp = xor1(std::string(pick, '0'), tmp);

return tmp;

}

// Function to validate that the string contains only '0' and '1'

bool isValidBinaryString(const string& str)

{

for (char c=str[0];c<str[str.length()-1];c++) {

if (c != '0' && c != '1')

return false;

}

return true;

}

// Function used at the sender side to encode data by appending the remainder of modular division at the end of data.

void encodeData(string data, string key)

{

int l\_key = key.length();

string appended\_data = (data + std::string(l\_key - 1, '0'));

string remainder = mod2div(appended\_data, key);

string codeword = data + remainder;

cout << "Remainder : " << remainder << "\n";

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

}

// Checking if the message received by the receiver is correct or not.

void receiver(string data, string key)

{

string currxor = mod2div(data.substr(0, key.size()), key);

int curr = key.size();

while (curr != data.size()) {

if (currxor.size() != key.size()) {

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

}

else {

currxor = mod2div(currxor, key);

}

}

if (currxor.size() == key.size()) {

currxor = mod2div(currxor, key);

}

if (currxor.find('1') != string::npos) {

cout << "There is some error in data" << endl;

}

else {

cout << "Correct message received" << endl;

}

}

int main()

{

string data, key;

// Input for data

cout << "Enter data string (only 0s and 1s): ";

cin >> data;

// Validate data input

while (!isValidBinaryString(data)) {

cout << "Invalid input. Please enter a binary string (only 0s and 1s): ";

cin >> data;

}

// Input for key

cout << "Enter key string (only 0s and 1s): ";

cin >> key;

// Validate key input

while (!isValidBinaryString(key)) {

cout << "Invalid input. Please enter a binary string (only 0s and 1s): ";

cin >> key;

}

cout << "\nSender side..." << endl;

encodeData(data, key);

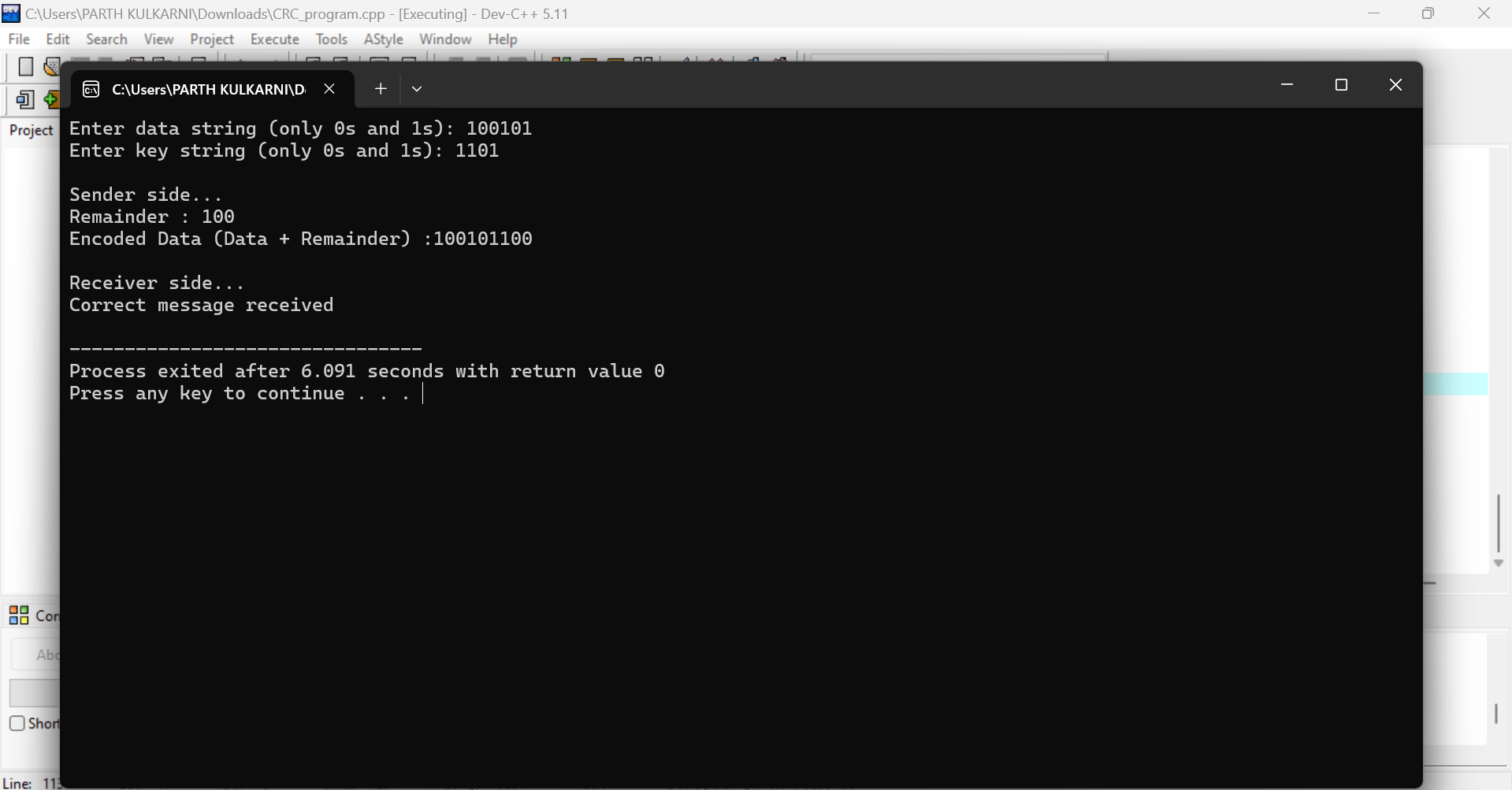
cout << "\nReceiver side..." << endl;

receiver(data + mod2div(data + std::string(key.size() - 1, '0'), key), key);

return 0;

}

**Output –**

****

1. **Hamming Code**

**Program –**

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

class hamming {

public:

string data; // raw data received

int m, r = 0; // m: length of raw data, r: count of redundant bits

char\* msg; // dynamic array to store data & redundant bits

hamming(string data) {

this->data = data;

// reversing the data received

reverse(data.begin(), data.end());

m = data.size();

int power = 1;

// finding the number of redundant bits and storing them in r

while (power < (m + r + 1)) {

r++;

power \*= 2;

}

// Allocating memory to our dynamic msg array

msg = new char[m + r + 1];

int curr = 0;

// initializing the msg with data bits and 'n' for redundant bits

for (int i = 1; i <= m + r; i++) {

if (i & (i - 1)) {

msg[i] = data[curr++];

}

else msg[i] = 'n';

}

// setting the redundant bits

setRedundantBits();

}

// function to show the whole msg

void showmsg() {

cout << "The data packet to be sent is: ";

for (int i = m + r; i >= 1; i--) {

cout << msg[i] << " ";

}

cout << endl;

}

// function to set the redundant bits

void setRedundantBits() {

int bit = 0;

for (int i = 1; i <= m + r; i \*= 2) {

int count = 0;

for (int j = i + 1; j <= m + r; j++) {

if (j & (1 << bit)) {

if (msg[j] == '1') count++;

}

}

// setting redundant bits

if (count & 1) msg[i] = '1';

else msg[i] = '0';

bit++;

}

showmsg();

}

// function to simulate the receiver

void receiver() {

string ans = "";

int bit = 0;

for (int i = 1; i <= m + r; i \*= 2) {

int count = 0;

for (int j = i + 1; j <= m + r; j++) {

if (j & (1 << bit)) {

if (msg[j] == '1') count++;

}

}

if (count & 1) {

if (msg[i] == '1') ans.push\_back('0');

else ans.push\_back('1');

}

else {

if (msg[i] == '0') ans.push\_back('0');

else ans.push\_back('1');

}

bit++;

}

if (ans.find('1') != string::npos) {

int power = 1;

int wrongbit = 0;

for (int i = 0; i < ans.size(); i++) {

if (ans[i] == '1') wrongbit += power;

power \*= 2;

}

cout << "Bit number " << wrongbit << " is wrong and has an error." << endl;

}

else {

cout << "Correct data packet received." << endl;

}

}

};

// Function to validate the binary string

bool isValidBinaryString(const string& str) {

for (char c=str[0];c<str[str.length()-1];c++) {

if (c != '0' && c != '1') return false;

}

return true;

}

int main() {

string data;

cout << "Enter data string (only 0s and 1s): ";

cin >> data;

// Validate the input string

while (!isValidBinaryString(data)) {

cout << "Invalid input. Please enter a binary string (only 0s and 1s): ";

cin >> data;

}

hamming h(data);

// Uncomment the line below to simulate an error by flipping a bit

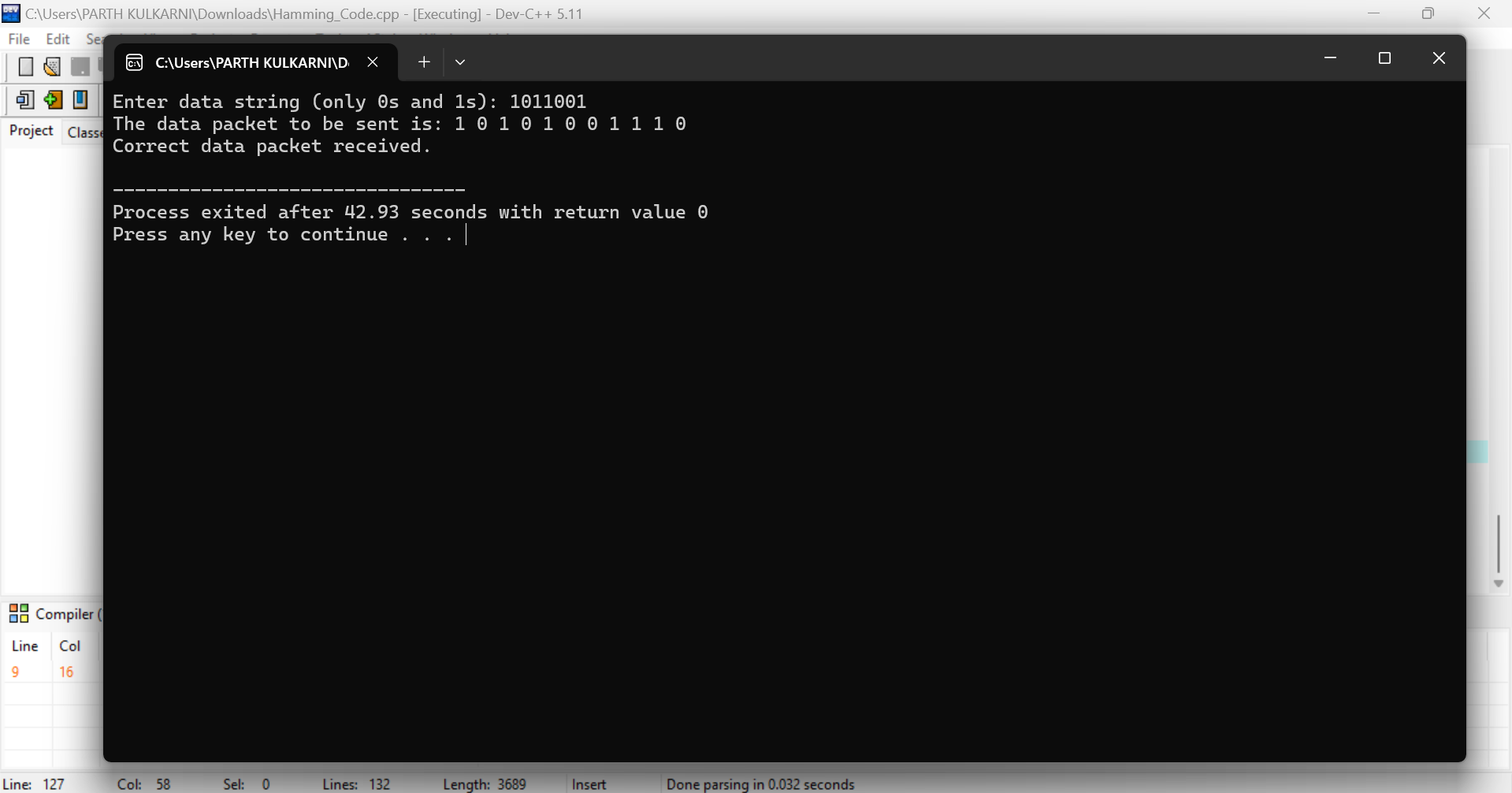
// h.msg[i] == '0' ? h.msg[i] = '1' : h.msg[i] = '0';

h.receiver();

return 0;

}

**Output –**

****