**Experiment No: 6 Date:**

**Error Detection**

**AIM:** Implement Error Detection Mechanisms using

1. Checksum
2. CRC

**THEORY:**

1. **Error Detection using Checksum in Computer Networks:**

* Error detection using checksum is a method employed in computer networks to ensure the integrity of transmitted data.
* It involves appending a checksum value to the data, which is calculated based on the content of the data itself.
* The receiver recalculates the checksum upon receiving the data and compares it with the transmitted checksum. If they match, the data is assumed to be intact; otherwise, errors are detected.

**Real-World Use Case:**

* Checksums are widely used in network protocols such as TCP/IP, UDP, and Ethernet.
* In TCP/IP, checksums are used at both the IP and TCP layers to ensure that data packets are transmitted without corruption.
* In Ethernet, a checksum called Frame Check Sequence (FCS) is used to detect errors in frames transmitted over the network.

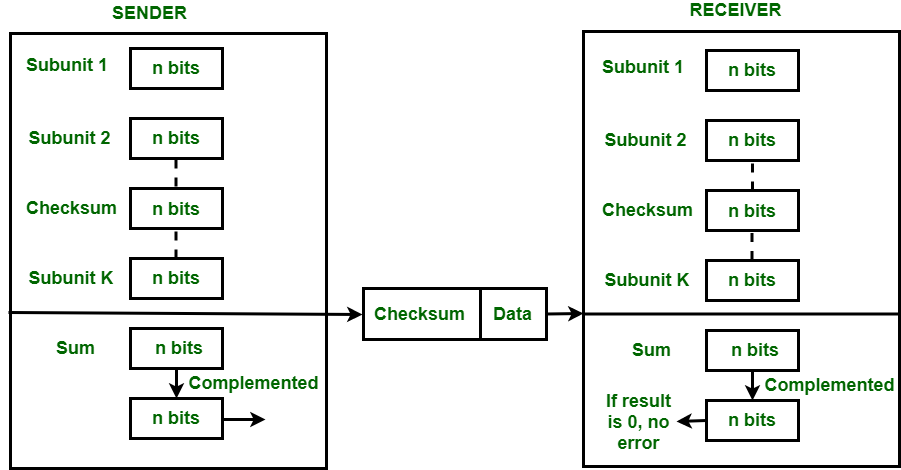
**Advantages:**

1. **Simple Implementation:** Checksum calculation and verification algorithms are relatively simple to implement, requiring minimal computational resources.
2. **Efficiency:** Checksums provide a lightweight mechanism for detecting errors without significant overhead.
3. **Error Detection:** Checksums are effective in detecting common transmission errors, such as single-bit errors or burst errors, enhancing the reliability of data transmission.

**Disadvantages:**

1. **Limited Error Detection Capability:** Checksums may not detect all types of errors, especially if multiple errors occur that cancel each other out or if the checksum length is insufficient.
2. **No Error Correction:** Checksums can only detect errors but cannot correct them. This limitation requires additional mechanisms, such as retransmission, to recover from errors.
3. **Vulnerability to Attacks:** Checksums alone are not secure against deliberate tampering or attacks, as an attacker could potentially modify both the data and the checksum to match.

**Error Detection using Checksum**

****

**Example:**

If the data unit to be transmitted is 10101001 00111001, the following procedure is used at Sender site and Receiver site.

**Sender Side:**

10101001 subunit 1

00111001 subunit 2

11100010 sum (using 1s complement)

**00011101**  checksum (complement of sum)

**Receiver Side:**

10101001 subunit 1

00111001 subunit 2

00011101 checksum

11111111 sum

**00000000** sum's complement

**Result is zero, it means no error.**

**CODE**

#include <iostream>

#include <string>

using namespace std;

int C = 0, b\_size;

string output;

char add(char a, char b) {

if (C == 0 && (a == '1' && b == '1')) //0 1 1{

C = 1;

return '0';

} else if (C == 0 && (a == '1' || b == '1')) //0 1 0 / 0 0 1{

C = 0;

return '1';

} else if (C == 0 && (a == '0' && b == '0')) //0 0 0{

C = 0;

return '0';

} else if (C == 1 && (a == '1' && b == '1')) //1 1 1{

C = 1;

return '1';

} else if (C == 1 && (a == '1' || b == '1')) //1 0 1 / 1 1 0{

C = 1;

return '0';

} else //1 0 0{

C = 0;

return '1';

}}

void complement() {

for (int k = b\_size - 1; k >= 0; k--) {

if (output[k] == '1')

output[k] = '0';

else

output[k] = '1';

}}

void compute\_sum(string data) {

string temp;

output = data.substr(0, b\_size);

for (int i = b\_size; i < data.length(); i += b\_size) {

int x = 0;

temp = data.substr(i, b\_size);

for (int k = b\_size - 1; k >= 0; k--)

output[k] = add(output[k], temp[k]);

if (C == 1) {

for (int k = b\_size - 1; k >= 0; k--) {

if (C == 1) {

C = 0;

output[k] = add(output[k], '1');

}}} }

complement();

}

int main() {

int ch, i, found;

string one,data;

while (1) {

cout << "\nChecksum\n1. Sender Side\n2. Receiver side\n3. Exit\nEnter choice: ";

cin >> ch;

switch (ch) {

case 1:

cout << "Enter data stream: ";

cin >> data;

cout << "Enter block size: ";

cin >> b\_size;

compute\_sum(data);

cout << "The checksum is: " << output << "\n";

// Append the checksum to the data

data.append(output);

cout << "The data with checksum is: " << data << "\n";

break;

case 2:

cout << "Enter data stream: ";

cin >> data;

cout << "Enter block size: ";

cin >> b\_size;

compute\_sum(data);

data = data.substr(0, data.length() - b\_size);

one = "1";

found = output.find(one);

if (found == string::npos) //0000

{

cout << "\nNo error present \nComplement of sum is " << output;

cout << "\nActual data is " << data << "\n";

} else //0010

{

cout << "Error present in code \n";

cout << "\nComplement of sum is " << output;

cout << "\nSince complement is not equal to 0 error is present\n\n";

}

break;

case 3:

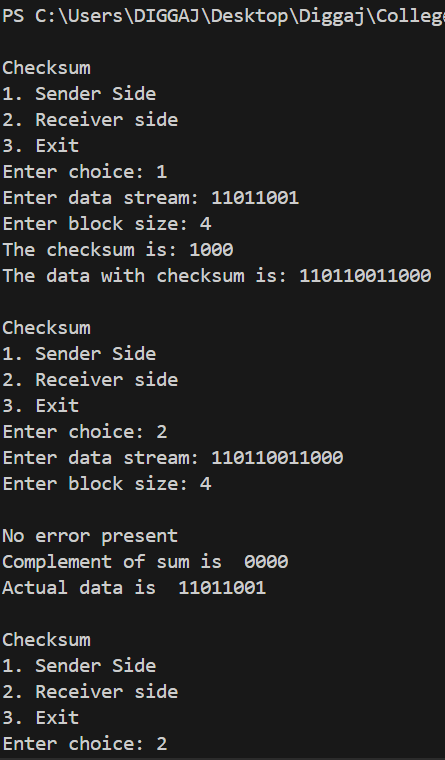
exit(0);

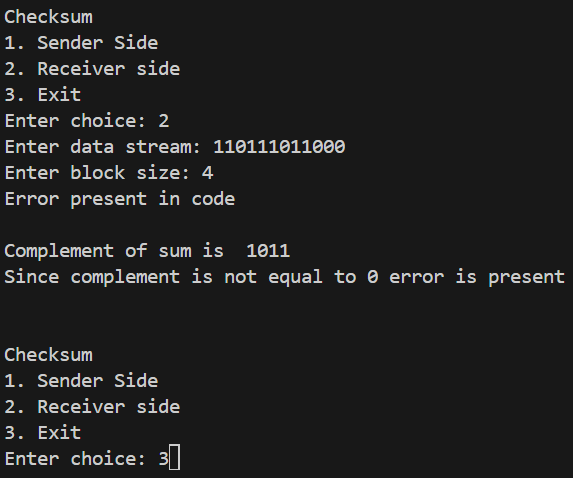
default:

cout << "Enter valid choice\n";

}}}

**OUTPUT**





1. **Error Detection using Cyclic Redundancy Check (CRC) in Computer Networks:**

**Introduction:**

* Cyclic Redundancy Check (CRC) is a method used for error detection in data transmission.
* CRC involves appending a checksum, computed using polynomial division, to the data being transmitted.
* At the receiver's end, the checksum is recalculated, and if it matches the received checksum, the data is considered intact; otherwise, errors are detected.

**Real-World Use Case:**

* CRC is widely employed in network protocols such as Ethernet, Wi-Fi, and Bluetooth.
* In Ethernet, CRC is used in the Frame Check Sequence (FCS) to detect errors in the transmitted frames.
* Similarly, in Wi-Fi and Bluetooth, CRC is utilized to ensure the integrity of data packets transmitted wirelessly.

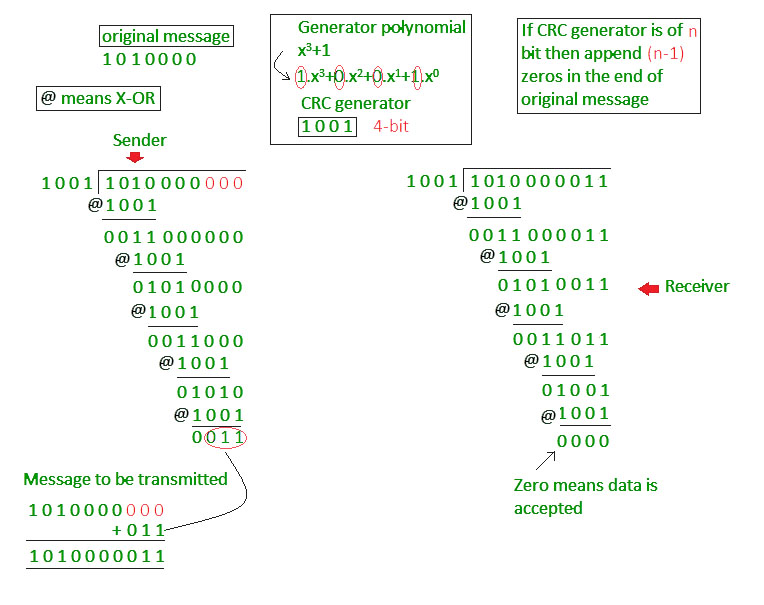
**Advantages:**

1. **High Accuracy:** CRC can detect a wide range of errors, including single-bit errors, burst errors, and most multiple-bit errors.
2. **Efficiency:** Despite its high error detection capability, CRC is computationally efficient, making it suitable for real-time data transmission.
3. **Error Localization:** CRC can not only detect errors but also provide some information about their location within the data packet, aiding in error correction strategies.

**Disadvantages:**

1. **Complexity:** Implementing CRC requires more computational resources and is more complex compared to simpler error detection techniques like checksums.
2. **False Positives:** Although rare, CRC can produce false positives, where the received data and checksum match even though errors are present, leading to undetected errors.
3. **Inability to Correct Errors:** Like checksums, CRC can only detect errors but cannot correct them, necessitating additional mechanisms for error recovery, such as retransmission.

**Example:**



**CODE**

#include<iostream>

using namespace std;

string 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 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;}

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";}

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;

cout << "Sender side" << endl;

cout << "Enter the Data" << endl;

cin >> data;

cout << "Enter the Generator" << endl;

cin >> key;

encodeData(data, key);

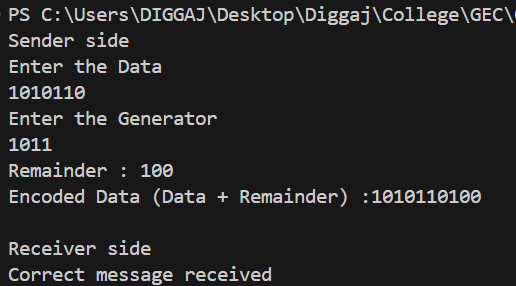
cout << "\nReceiver side" << endl;

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

return 0;

}

**OUTPUT**



**CONCLUSION:**

The Error Detection using Checksum and Cyclic Redundancy Check (CRC) was studied and implemented successfully.