Experiment 8:

* Aim : Cyclic redundancy check  In c
* Apparatus (Software):
* Turbo C++

**Algorithm:**

**Sender Side (CRC Generation)**

1. **Input**:
   * The data bits to be transmitted (data).
   * The CRC polynomial (polynomial).
2. **Augment the data**:
   * Append n-1 zeros to the data, where n is the degree of the CRC polynomial.
3. **Perform Modulo-2 Division (XOR Operation)**:
   * Divide the augmented data by the polynomial using modulo-2 division (XOR operation).
   * Shift left the data and perform XOR with the polynomial if the leading bit is 1.
   * Obtain the remainder as the CRC checksum.
4. **Append the CRC Checksum**:
   * Append the CRC checksum (remainder) to the original data.
5. **Transmit the augmented data**.

**Receiver Side (CRC Error Detection)**

1. **Input**:
   * The received data (including CRC checksum) and the polynomial.
2. **Perform Modulo-2 Division (XOR Operation)**:
   * Divide the received data by the polynomial using modulo-2 division (XOR operation).
   * Shift left and perform XOR as required to get the remainder.
3. **Check the remainder**:
   * If the remainder is all zeros, the data is error-free.
   * Otherwise, an error is detected.
4. **End**.

Code:

#include <stdio.h>

#include <conio.h>

#include <string.h>

void xorOperation(char dividend[], char divisor[], int divisor\_len);

void crcDivision(char data[], char generator[], char remainder[], int data\_len, int gen\_len);

void generateCRC(char data[], char generator[], int data\_len, int gen\_len);

int checkCRC(char data[], char generator[], int data\_len, int gen\_len);

void xorOperation(char dividend[], char divisor[], int divisor\_len) {

int i;

for (i = 0; i < divisor\_len; i++) {

dividend[i] = (dividend[i] == divisor[i]) ? '0' : '1';

}

}

void crcDivision(char data[], char generator[], char remainder[], int data\_len, int gen\_len) {

int i, j;

for (i = 0; i < gen\_len; i++) {

remainder[i] = data[i];

}

for (i = 0; i < data\_len - gen\_len + 1; i++) {

if (remainder[0] == '1') {

xorOperation(remainder, generator, gen\_len);

}

for (j = 0; j < gen\_len - 1; j++) {

remainder[j] = remainder[j + 1];

}

remainder[gen\_len - 1] = data[gen\_len + i];

}

}

void generateCRC(char data[], char generator[], int data\_len, int gen\_len) {

char remainder[32];

int i;

crcDivision(data, generator, remainder, data\_len + gen\_len - 1, gen\_len);

for (i = 0; i < gen\_len - 1; i++) {

data[data\_len + i] = remainder[i];

}

data[data\_len + gen\_len - 1] = '\0';

}

int checkCRC(char data[], char generator[], int data\_len, int gen\_len) {

char remainder[32];

int i;

crcDivision(data, generator, remainder, data\_len, gen\_len);

for (i = 0; i < gen\_len - 1; i++) {

if (remainder[i] != '0') {

return 0; // Error detected

}

}

return 1;

}

int main() {

char data[64], generator[32];

int data\_len, gen\_len, i;

clrscr();

printf("Enter the data bits: ");

scanf("%s", data);

printf("Enter the CRC polynomial: ");

scanf("%s", generator);

data\_len = strlen(data);

gen\_len = strlen(generator);

for (i = 0; i < gen\_len - 1; i++) {

data[data\_len + i] = '0';

}

data[data\_len + gen\_len - 1] = '\0';

printf("Original Data (with zeros): %s\n", data);

generateCRC(data, generator, data\_len, gen\_len);

printf("Data with CRC Checksum: %s\n", data);

if (checkCRC(data, generator, data\_len + gen\_len - 1, gen\_len)) {

printf("No error detected in received data.\n");

} else {

printf("Error detected in received data.\n");

}

getch();

return 0;

}

Output:

