Terna Engineering College

Computer Engineering Department

Program: Sem V

PART A

(PART A: TO BE COMPLETED BY STUDENTS)

Experiment No.6

A.1 Aim:

M

A.2 Objective:

Implementation of a Cyclic Redundancy Code (CRC) generator and checker using any higher level language

A.3 Prerequisite:

- Knowledge about PAN, LAN and NW Elements.
- Knowledge of Programming Languages.
- Binary arithmetic's.
- Error types and their detection and correction.
- Concept of Programming, Analysis, Design, Simulation and Modelling

A.4 Outcome:

After successful completion of this experiment students will be able to

- Ability to select the proper NW Elements required to design NWs.
- Thorough understanding of DLL.
- Error detection methodologies and their implementation.
- Hard coding by applying their programming skills.

A.5 Theory/Tutorial:

The Cyclic Redundancy Check algorithm checks for errors and verifies the accuracy of the data delivered by the sender. CRC requires a generator polynomial in order to compute the check value using binary division in addition to the data that has to be transferred. To ensure that the data is genuine, the check value or CRC is sent with it to the recipient.

The degree of the polynomial can be used as the bit locations to represent the data that will be conveyed to the recipient in polynomial form.

Binary data can also be used to represent the generating polynomial. The degree of the data polynomial must be less than the degree of the generator polynomial and must be larger than 0. Based on the degree of the generating polynomial, the CRC may be divided into several standards. Using a generator polynomial of degree 8 for the CRC-8 standard and degree 16 for the CRC-16 standard.

An example generator polynomial is of the form like $x^3 + x + 1$. This generator polynomial represents key 1011. Another example is $x^2 + 1$ that represents key 101.

n : Number of bits in data to be sent from sender side. k : Number of bits in the key obtained

from generator polynomial.

Sender Side (Generation of Encoded Data from Data and Generator Polynomial (or Key)):

1. The binary data is first augmented by adding k-1 zeros in the end of the data
2. Use **modulo-2 binary division** to divide binary data by the key and store remainder of division.
3. Append the remainder at the end of the data to form the encoded data and send the same

Receiver Side (Check if there are errors introduced in transmission)
Perform modulo-2 division again and if the remainder is 0, then there are no errors.

Modulo 2 Division:
The process of modulo-2 binary division is the same as the familiar division process we use for decimal numbers. Just that instead of subtraction, we use XOR here.

- In each step, a copy of the divisor (or data) is XORed with the k bits of the dividend (or key).
- The result of the XOR operation (remainder) is (n-1) bits, which is used for the next step after 1 extra bit is pulled down to make it n bits long.
- When there are no bits left to pull down, we have a result. The (n-1)-bit remainder which is appended at the sender side

References:

- https://www.javatpoint.com/computer-network-error-detection

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

Roll No. A11	Name: Khan Mohammad TAQI Karrar Husain
Class : T.E A	Batch : A1
Date of Experiment:	Date of Submission
Grade :	

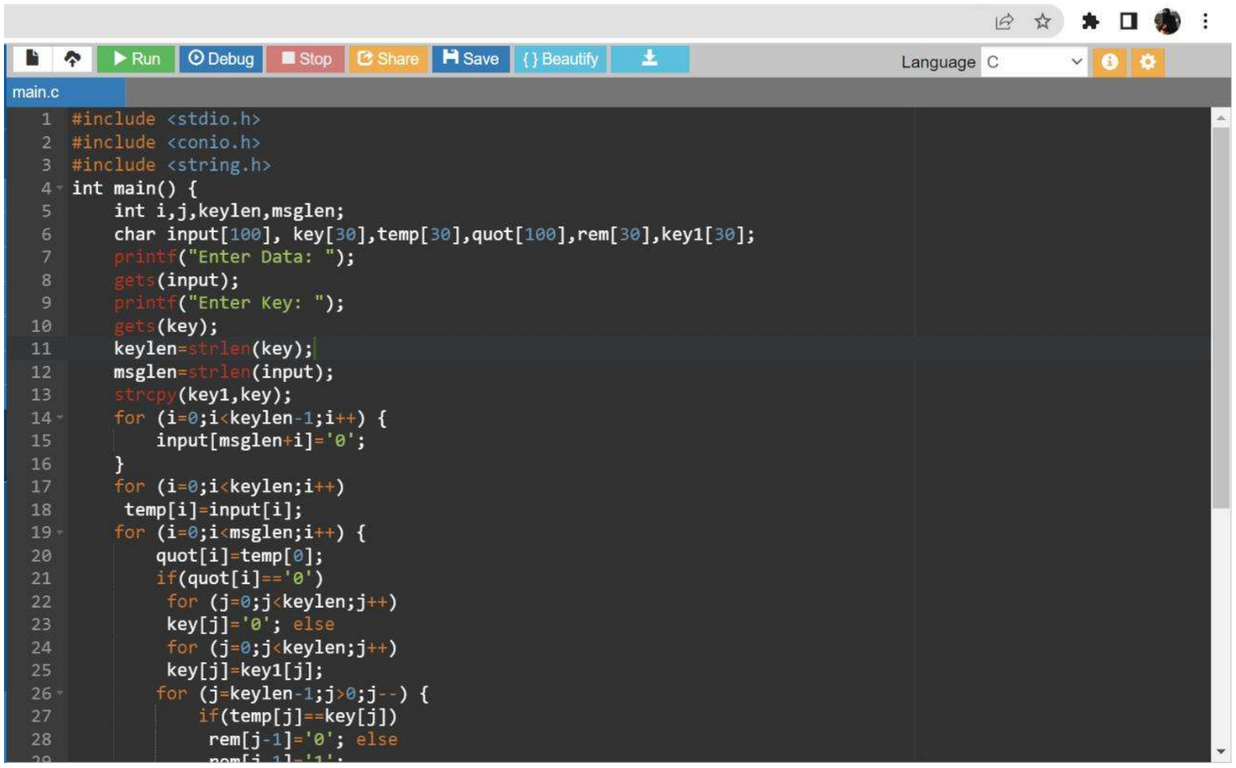
B.1 Document created by the student:

CRC Program in C

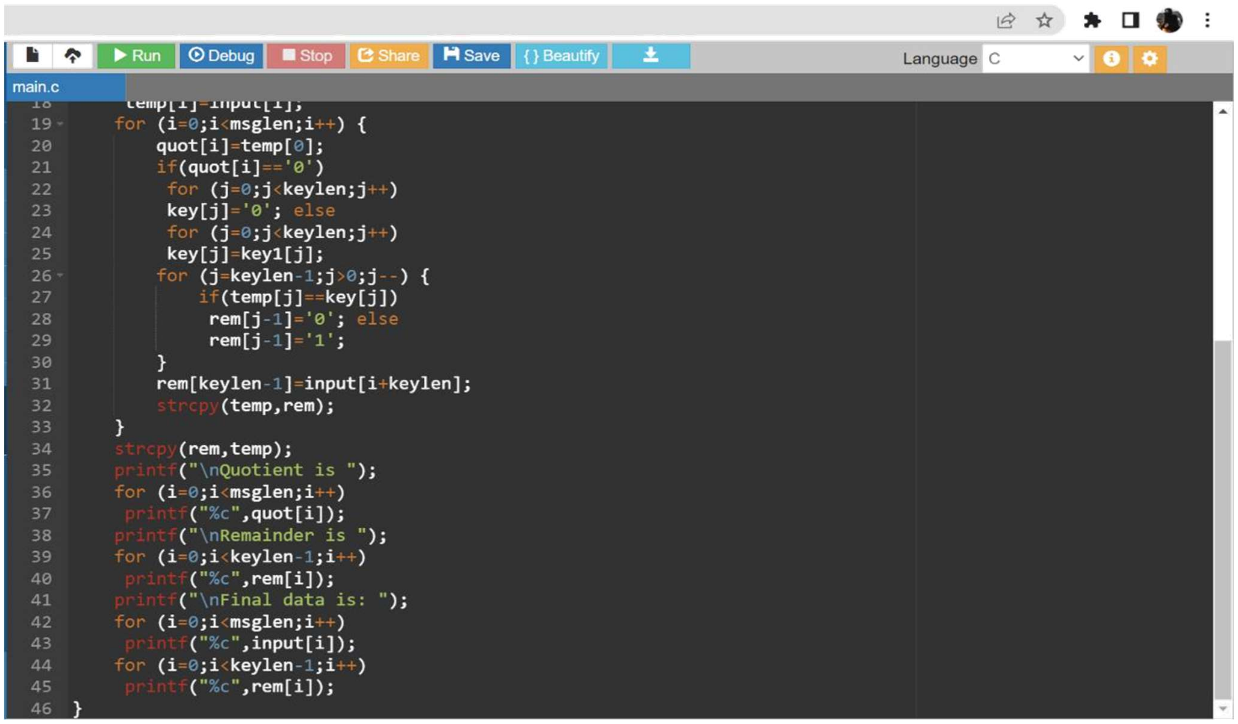
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main() {
        int i,j,keylen,msglen;
        char input[100], key[30],temp[30],quot[100],rem[30],key1[30];
        printf("Enter Data: ");
        gets(input);
        printf("Enter Key: ");
        gets(key);
        keylen=strlen(key);
        msglen=strlen(input);
        strcpy(key1,key);
        for (i=0;i<keylen-1;i++) {
                input[msglen+i]='0';
        }
        for (i=0;i<keylen;i++)
         temp[i]=input[i];
        for (i=0;i<msglen;i++) {
                quot[i]=temp[0];
                if(quot[i]=='0')
                 for (j=0;j<keylen;j++)
                 key[j]='0'; else
                 for (j=0;j<keylen;j++)
                 key[j]=key1[j];
                for (j=keylen-1;j>0;j--) {
                      if(temp[j]==key[j])
                       rem[j-1]='0'; else
                       rem[j-1]='1';
                }
                rem[keylen-1]=input[i+keylen];
                strcpy(temp,rem);
        }
        strcpy(rem,temp);
        printf("\nQuotient is ");
        for (i=0;i<msglen;i++)
         printf("%c",quot[i]);
        printf("\nRemainder is ");
        for (i=0;i<keylen-1;i++)
         printf("%c",rem[i]);
        printf("\nFinal data is: ");
        for (i=0;i<msglen;i++)
         printf("%c",input[i]);
        for (i=0;i<keylen-1;i++)
         printf("%c",rem[i]);
}
```

B.2 Input and Output:

Input:

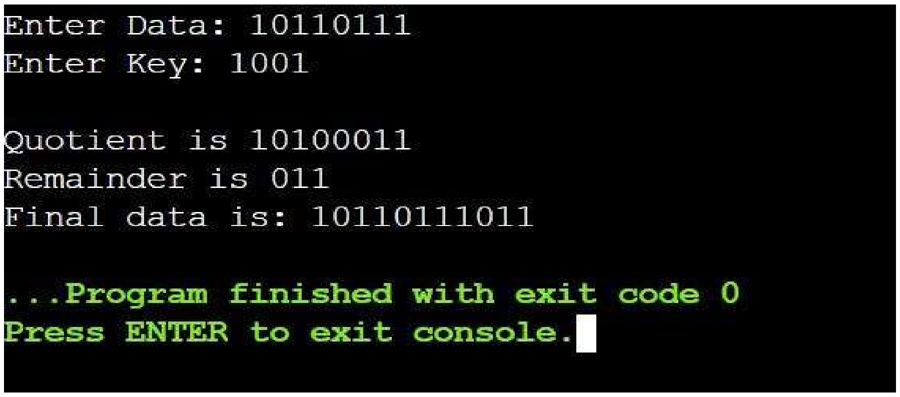


```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main() {
    int i,j,keylen,msglen;
    char input[100], key[30],temp[30],quot[100],rem[30],key1[30];
    printf("Enter Data: ");
    gets(input);
    printf("Enter Key: ");
    gets(key);
    keylen=strlen(key);
    msglen=strlen(input);
    strcpy(key1,key);
    for (i=0;i<keylen-1;i++) {
        input[msglen+i]='0';
    }
    for (i=0;i<keylen;i++)
     temp[i]=input[i];
    for (i=0;i<msglen;i++) {
        quot[i]=temp[0];
        if(quot[i]=='0')
         for (j=0;j<keylen;j++)
         key[j]='0'; else
         for (j=0;j<keylen;j++)
         key[j]=key1[j];
        for (j=keylen-1;j>0;j--) {
            if(temp[j]==key[j])
             rem[j-1]='0'; else
```



```c
        for (i=0;i<msglen;i++) {
            quot[i]=temp[0];
            if(quot[i]=='0')
             for (j=0;j<keylen;j++)
             key[j]='0'; else
             for (j=0;j<keylen;j++)
             key[j]=key1[j];
            for (j=keylen-1;j>0;j--) {
                if(temp[j]==key[j])
                 rem[j-1]='0'; else
                 rem[j-1]='1';
            }
            rem[keylen-1]=input[i+keylen];
            strcpy(temp,rem);
        }
        strcpy(rem,temp);
        printf("\nQuotient is ");
        for (i=0;i<msglen;i++)
         printf("%c",quot[i]);
        printf("\nRemainder is ");
        for (i=0;i<keylen-1;i++)
         printf("%c",rem[i]);
        printf("\nFinal data is: ");
        for (i=0;i<msglen;i++)
         printf("%c",input[i]);
        for (i=0;i<keylen-1;i++)
         printf("%c",rem[i]);
    }
```

Output:



```
Enter Data: 10110111
Enter Key: 1001

Quotient is 10100011
Remainder is 011
Final data is: 10110111011

...Program finished with exit code 0
Press ENTER to exit console.
```

B.3 Observations and learning:

We have learned how to generate Cyclic Redundancy Code (CRC) and to check error bits in data by using C language.

B.4 Conclusion:

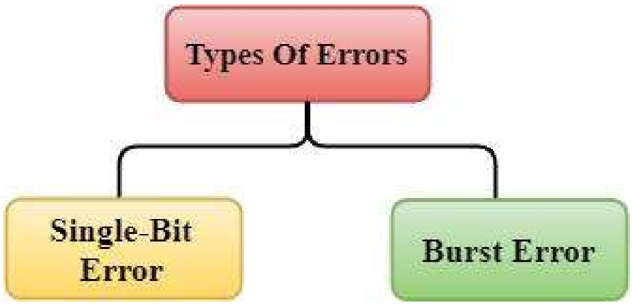
We have successfully implemented the Cyclic Redundancy Code (CRC) generator and checker using C language.

B.5 Question of Curiosity:

1. What is an error? Name the types of error?

Ans.: When data is transmitted from one device to another device, the system does not guarantee whether the data received by the device is identical to the data transmitted by another device. An Error is a situation when the message received at the receiver end is not identical to the message transmitted.

Types Of Errors



Errors can be classified into two categories:

- Single-Bit Error
- Burst Error

2. Single bit error is found in parallel transmission. Give valid reason.

Ans.: Single bit errors are the least likely type of errors in serial data transmissionbecause **the noise must have a very short duration** which is very rare. However, this kind of errors can happen in parallel transmission. Example: If data is sent at 1Mbps then each bit lasts only 1/1,000,000 sec. or 1 μs. Single-Bit Error mainly occurs in Parallel Data Transmission. For example, if eight wires are used to send the eight bits of a byte, if one of the wire is noisy, then single-bit is corrupted per byte.

3. Burst error is normally found in serial transmission. Give reason.

Ans.: Burst error is most likely to happen in serial transmission since **the duration of noise is normally longer than the duration of a bit**. The number of bits affected depends on the data rate and duration of noise.

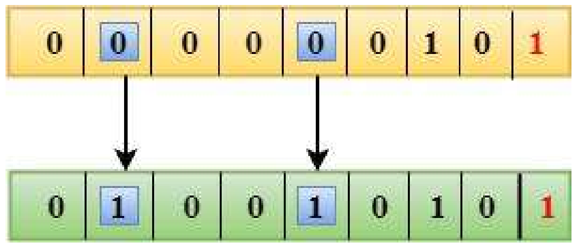
4. What are even and odd parity? State the limitation of Single parity check and Two-dimensional parity check.

Ans.: There are two kinds of parity bits:

- **In even parity**, the number of bits with a value of one are counted. If that number is odd, the parity bit value is set to one to make the total number of ones in the set (including the parity bit) an even number. If the number of bits with a value of one is even, the parity bit value is set to zero, so that the total number of ones in the set (including the parity bit) remains an even number.
- **In odd parity**, if the number of bits with a value of one is an even number, the parity bit value is set to one to make the total number of ones in the set (including the parity bit) an odd number. If the number of bits with a value of one is odd, the parity bit value is set to zero, so that the total number of ones in the set (including the parity bit) remains an odd number.
- **Limitations Of Single Parity Checking**

o It can only detect single-bit errors which are very rare.

o If two bits are interchanged, then it cannot detect the errors.



Limitations Of Two-dimensional Parity Check

o If two bits in one data unit are corrupted and two bits the same position in another data unit are also corrupted, then 2D Parity checker will not be able to detect the error.

o This technique cannot be used to detect the 4-bit errors or more in some cases.

5. What are the redundant bit Generator and error Checker?

Ans.: Redundant bits are the bits that is generated to be append on the data on transmission side and that transmitted data is checked on receiver side to detect error in transmitted data. VRC, CRC and LRC generator is the redundant bit generator and VRC, CRC and LRC checker is the redundant bit checker.

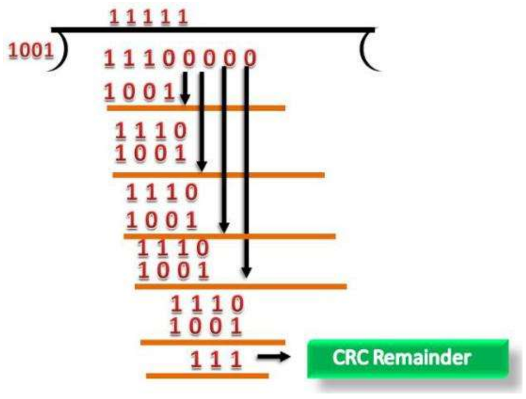
6. State the working of CRC error detection method with example.

Ans.:

CRC Generator

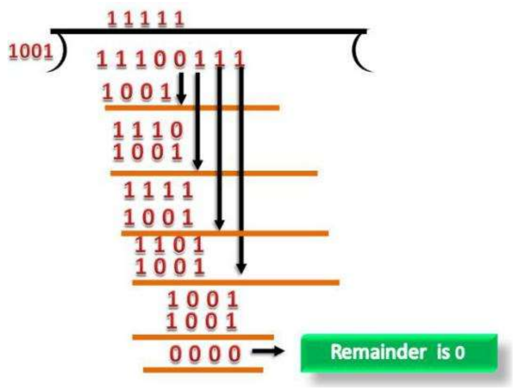
- A CRC generator uses a modulo-2 division. Firstly, three zeroes are appended at the end of the data as the length of the divisor is 4 and we know that the length of the string 0s to be appended is always one less than the length of the divisor.
- Now, the string becomes 11100000, and the resultant string is divided by the divisor 1001.
- The remainder generated from the binary division is known as CRC remainder. The generated value of the CRC remainder is 111.
- CRC remainder replaces the appended string of 0s at the end of the data unit, and the final string would be 11100111 which is sent across the network.

p



CRC Checker

- The functionality of the CRC checker is similar to the CRC generator.
- When the string 11100111 is received at the receiving end, then CRC checker performs the modulo-2 division.
- A string is divided by the same divisor, i.e., 1001.
- In this case, CRC checker generates the remainder of zero. Therefore, the data is accepted.



7. Which method is used for Forward error correction?

Ans.: Forward error correction (FEC) is an error correction technique to detect and correct a limited number of errors in transmitted data without the need for retransmission. In this method, the sender sends a redundant error-correcting code along with the data frame. The receiver performs necessary checks based upon the additional redundant bits. If it finds that the data is free from errors, it executes error-correcting code that generates the actual frame. It then removes the redundant bits before passing the message to the upper layers.