Deepak R **18ETCS002041**

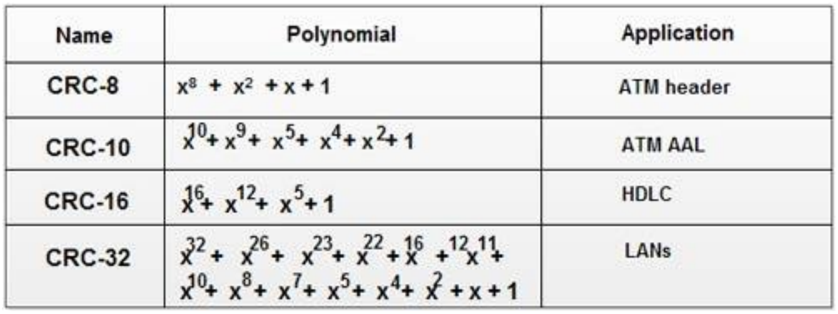
Experiment 2: Error Detection using CRC-CCITT

Aim: To apply CRC (CCITT Polynomial) for error detection

Objective: After carrying out this experiment, students will be able to:

- Apply CRC CCITT to develop codes for error detection
- Analyse how this CRC is able to detect bit errors irrespective of their length and position in the data

Problem statement: You are required to write a program that uses CRC to detect burst errors in transmitted data. Initially, write the program using the CRC example you studied in class. Your final program should ask the user to input data and choose a generator polynomial from the list given in the figure below. Your program is required to calculate the checksum and the transmitted data. Subsequently, the user enters the received data. Applying the same generator polynomial on the received data should result in a remainder of 0.



Name	Polynomial	Application
CRC-8	$x^8 + x^2 + x + 1$	ATM header
CRC-10	$x^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$x^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Analysis: While analyzing your program, you are required to address the following points:

- How is this method different from 2D parity scheme that you have implemented previously?
- What are the limitations of this method of error detection?

MARKS DISTRIBUTION

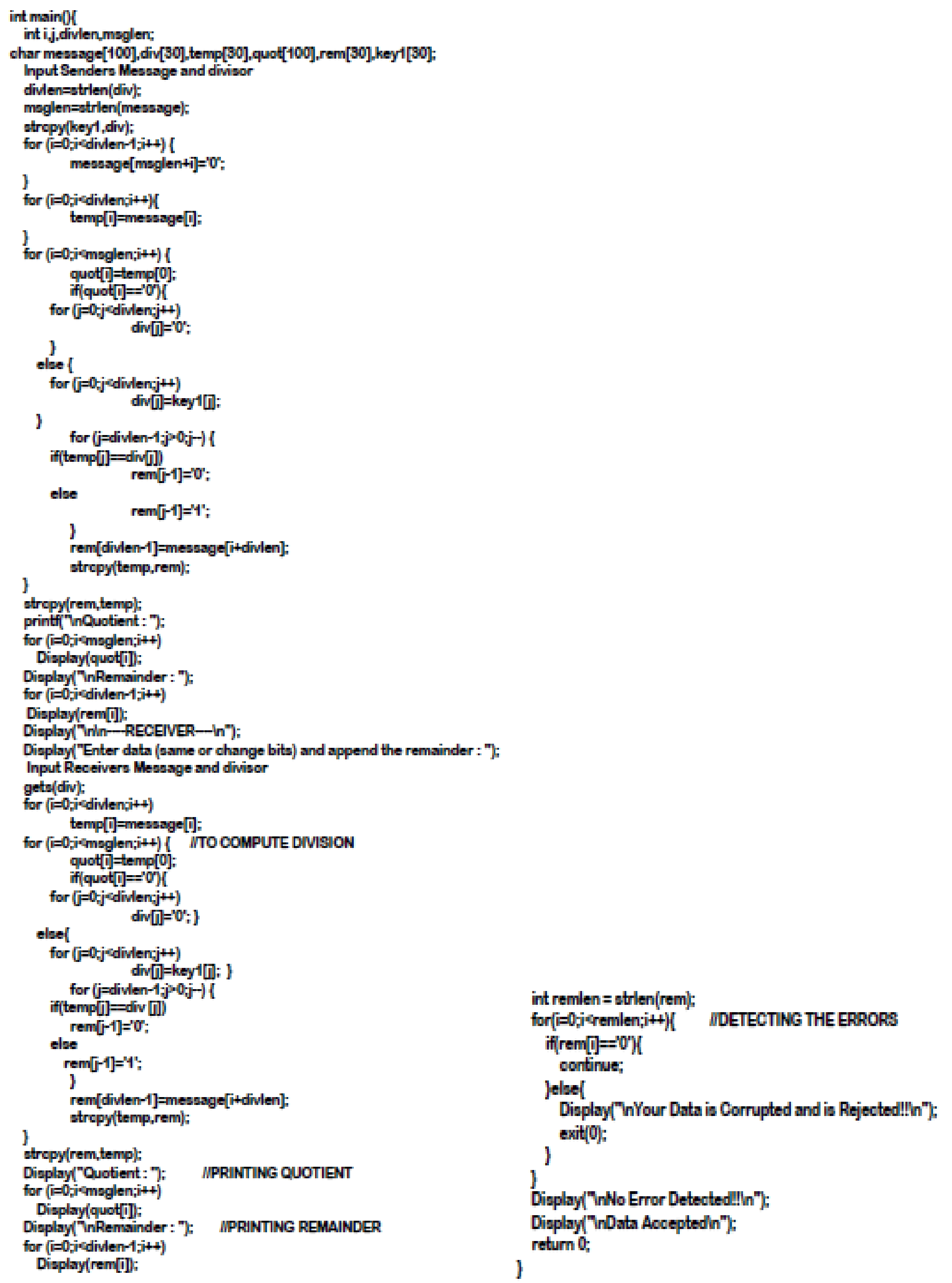
Component	Maximum Marks	Marks Obtained
Preparation of Document	7	
Results	7	
Viva	6	
Total	**20**	

Submitted by: Deepak R
Register No: 18ETCS002041

Ramaiah University of Applied Sciences

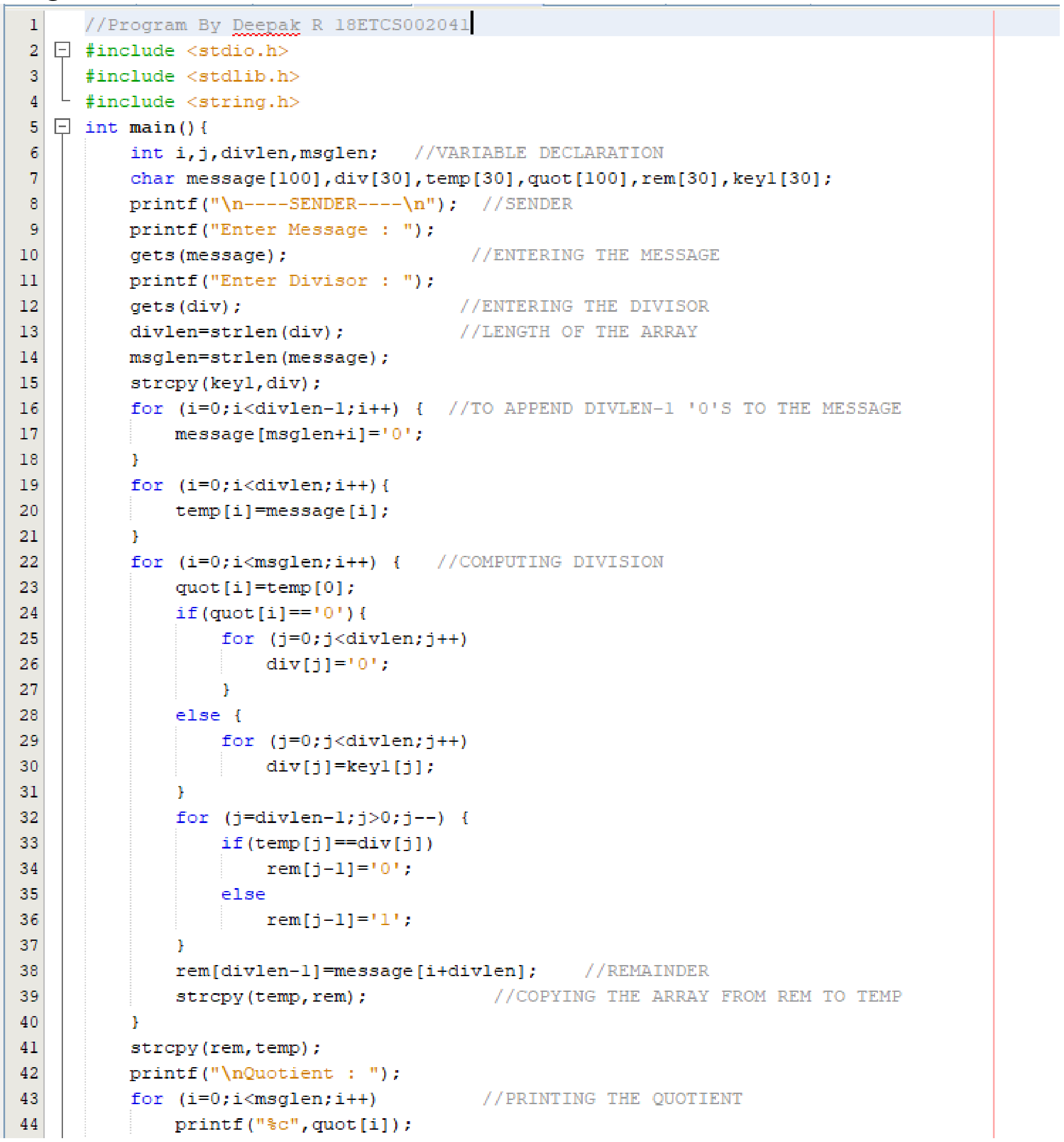
1. Algorithm/Flowchart

Pseudocode for Given Program

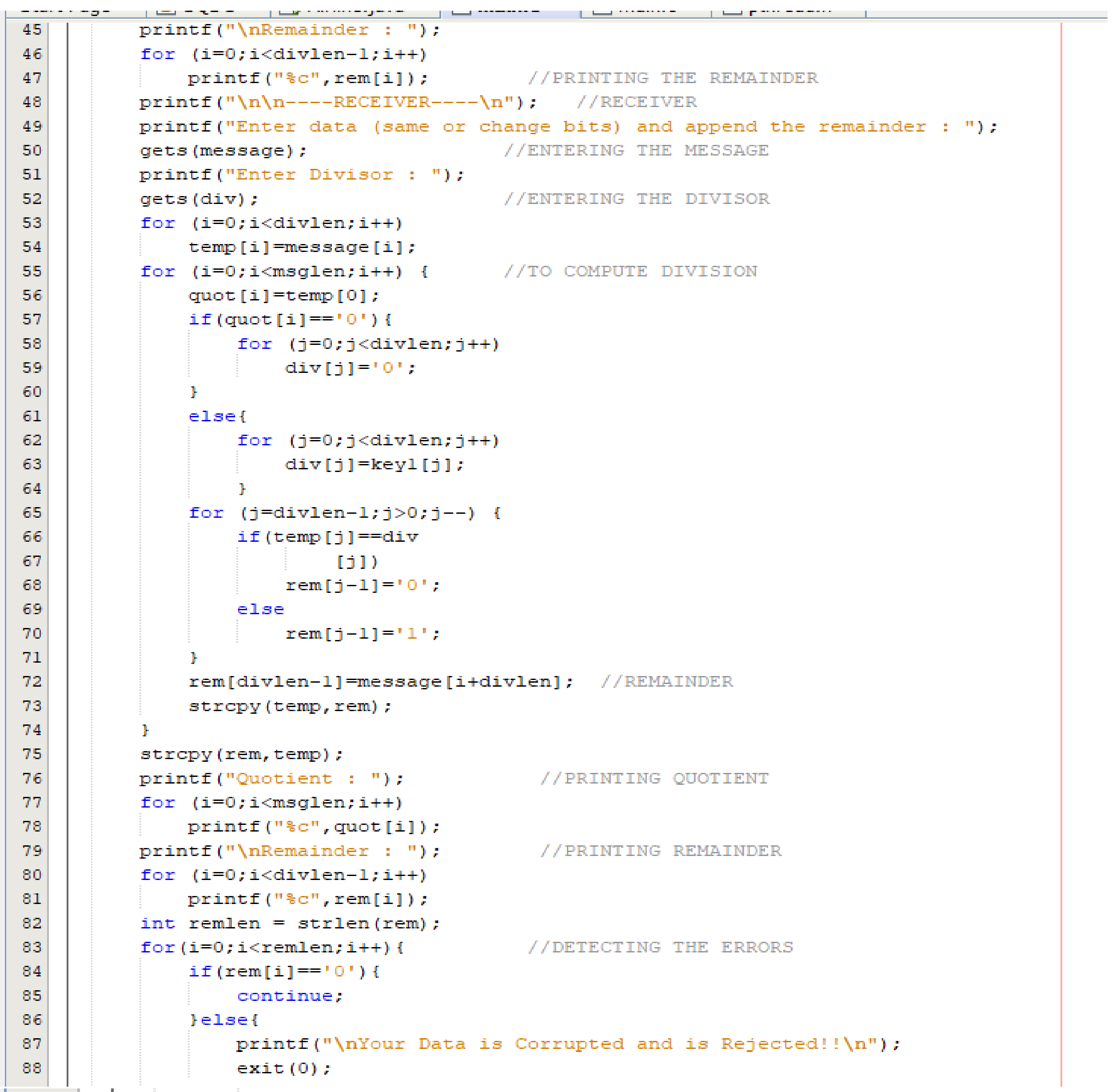


```
int main(){
   int i,j,divlen,msglen;
char message[100],div[30],temp[30],quot[100],rem[30],key1[30];
   Input Senders Message and divisor
   divlen=strlen(div);
   msglen=strlen(message);
   strcpy(key1,div);
   for (i=0;i<divlen-1;i++) {
            message[msglen+i]='0';
   }
   for (i=0;i<divlen;i++){
            temp[i]=message[i];
   }
   for (i=0;i<msglen;i++) {
            quot[i]=temp[0];
            if(quot[i]=='0'){
        for (j=0;j<divlen;j++)
                        div[j]='0';
        }
      else {
        for (j=0;j<divlen;j++)
                        div[j]=key1[j];
      }
            for (j=divlen-1;j>0;j--) {
        if(temp[j]==div[j])
                        rem[j-1]='0';
        else
                        rem[j-1]='1';
            }
            rem[divlen-1]=message[i+divlen];
            strcpy(temp,rem);
   }
   strcpy(rem,temp);
   printf("\nQuotient : ");
   for (i=0;i<msglen;i++)
      Display(quot[i]);
   Display("\nRemainder : ");
   for (i=0;i<divlen-1;i++)
    Display(rem[i]);
   Display("\n\n----RECEIVER----\n");
   Display("Enter data (same or change bits) and append the remainder : ");
    Input Receivers Message and divisor
   gets(div);
   for (i=0;i<divlen;i++)
            temp[i]=message[i];
   for (i=0;i<msglen;i++) {      //TO COMPUTE DIVISION
            quot[i]=temp[0];
            if(quot[i]=='0'){
        for (j=0;j<divlen;j++)
                        div[j]='0'; }
      else{
        for (j=0;j<divlen;j++)
                        div[j]=key1[j];  }
            for (j=divlen-1;j>0;j--) {
        if(temp[j]==div [j])
            rem[j-1]='0';
        else
          rem[j-1]='1';
            }
            rem[divlen-1]=message[i+divlen];
            strcpy(temp,rem);
   }
   strcpy(rem,temp);
   Display("Quotient : ");        //PRINTING QUOTIENT
   for (i=0;i<msglen;i++)
      Display(quot[i]);
   Display("\nRemainder : ");      //PRINTING REMAINDER
   for (i=0;i<divlen-1;i++)
      Display(rem[i]);
   int remlen = strlen(rem);
   for(i=0;i<remlen;i++){        //DETECTING THE ERRORS
     if(rem[i]=='0'){
        continue;
     }else{
        Display("\nYour Data is Corrupted and is Rejected!!\n");
        exit(0);
     }
   }
   Display("\nNo Error Detected!!\n");
   Display("\nData Accepted\n");
   return 0;
}
```

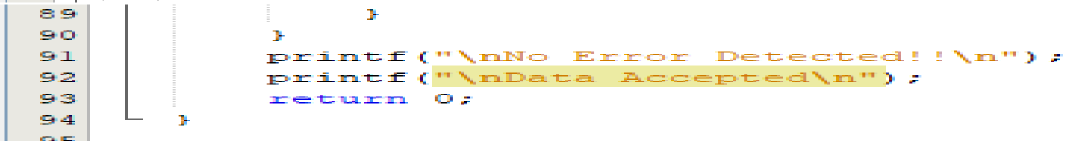
2. Program



```c
//Program By Deepak R 18ETCS002041
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(){
    int i,j,divlen,msglen;   //VARIABLE DECLARATION
    char message[100],div[30],temp[30],quot[100],rem[30],key1[30];
    printf("\n----SENDER----\n");  //SENDER
    printf("Enter Message : ");
    gets(message);                  //ENTERING THE MESSAGE
    printf("Enter Divisor : ");
    gets(div);                      //ENTERING THE DIVISOR
    divlen=strlen(div);             //LENGTH OF THE ARRAY
    msglen=strlen(message);
    strcpy(key1,div);
    for (i=0;i<divlen-1;i++) {  //TO APPEND DIVLEN-1 '0'S TO THE MESSAGE
        message[msglen+i]='0';
    }
    for (i=0;i<divlen;i++){
        temp[i]=message[i];
    }
    for (i=0;i<msglen;i++) {   //COMPUTING DIVISION
        quot[i]=temp[0];
        if(quot[i]=='0'){
            for (j=0;j<divlen;j++)
                div[j]='0';
            }
        else {
            for (j=0;j<divlen;j++)
                div[j]=key1[j];
        }
        for (j=divlen-1;j>0;j--) {
            if(temp[j]==div[j])
                rem[j-1]='0';
            else
                rem[j-1]='1';
        }
        rem[divlen-1]=message[i+divlen];    //REMAINDER
        strcpy(temp,rem);               //COPYING THE ARRAY FROM REM TO TEMP
    }
    strcpy(rem,temp);
    printf("\nQuotient : ");
    for (i=0;i<msglen;i++)          //PRINTING THE QUOTIENT
        printf("%c",quot[i]);
```

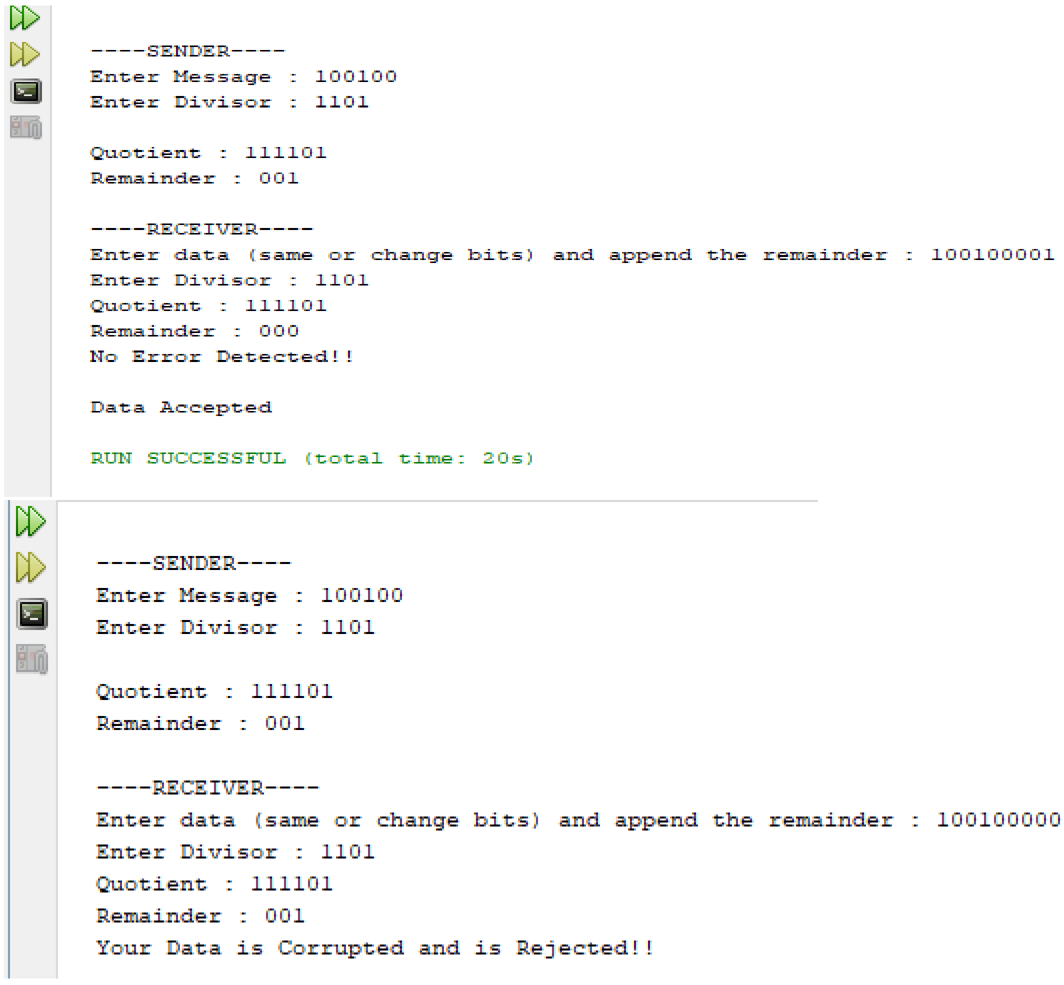


```
    printf("\nRemainder : ");
    for (i=0;i<divlen-1;i++)
        printf("%c",rem[i]);         //PRINTING THE REMAINDER
    printf("\n\n----RECEIVER----\n");   //RECEIVER
    printf("Enter data (same or change bits) and append the remainder : ");
    gets(message);                   //ENTERING THE MESSAGE
    printf("Enter Divisor : ");
    gets(div);                       //ENTERING THE DIVISOR
    for (i=0;i<divlen;i++)
        temp[i]=message[i];
    for (i=0;i<msglen;i++) {         //TO COMPUTE DIVISION
        quot[i]=temp[0];
        if(quot[i]=='0'){
            for (j=0;j<divlen;j++)
                div[j]='0';
        }
        else{
            for (j=0;j<divlen;j++)
                div[j]=key1[j];
            }
        for (j=divlen-1;j>0;j--) {
            if(temp[j]==div
                    [j])
                rem[j-1]='0';
            else
                rem[j-1]='1';
        }
        rem[divlen-1]=message[i+divlen];   //REMAINDER
        strcpy(temp,rem);
    }
    strcpy(rem,temp);
    printf("Quotient : ");             //PRINTING QUOTIENT
    for (i=0;i<msglen;i++)
        printf("%c",quot[i]);
    printf("\nRemainder : ");          //PRINTING REMAINDER
    for (i=0;i<divlen-1;i++)
        printf("%c",rem[i]);
    int remlen = strlen(rem);
    for(i=0;i<remlen;i++){           //DETECTING THE ERRORS
        if(rem[i]=='0'){
            continue;
        }else{
            printf("\nYour Data is Corrupted and is Rejected!!\n");
            exit(0);
```



```
        }
    }
    printf("\nNo Error Detected!!\n");
    printf("\nData Accepted\n");
    return 0;
}
```

3. Results



```
----SENDER----
Enter Message : 100100
Enter Divisor : 1101

Quotient : 111101
Remainder : 001

----RECEIVER----
Enter data (same or change bits) and append the remainder : 100100001
Enter Divisor : 1101
Quotient : 111101
Remainder : 000
No Error Detected!!

Data Accepted

RUN SUCCESSFUL (total time: 20s)
```

```
----SENDER----
Enter Message : 100100
Enter Divisor : 1101

Quotient : 111101
Remainder : 001

----RECEIVER----
Enter data (same or change bits) and append the remainder : 100100000
Enter Divisor : 1101
Quotient : 111101
Remainder : 001
Your Data is Corrupted and is Rejected!!
```

Figure 0-1 OUTPUT

Explanation for Figure 0-1 OUTPUT:

Here the data bits are taken as 100100, and for which divisor is 1101, performing manual CRC calculation resulted in the CRC to be 100100001, which is same as the output from the program, the message is then padded with the CRC at the end and the received message is given to the program which then verifies the output. Printing PASS if the message is error free, FAIL otherwise. Following the output, is the input for the same message but the message that is received is altered, here the error is detected since the remainder from the message is non-zero.

4. Analysis and Discussions

The conventional method also generally utilizes cyclic redundancy check (CRC) bits for error detection purposes. In particular, a fixed number of CRC bits are appended to the end of each message block and have a predetermined relationship with the corresponding message block. The receiver receives both the message block and the CRC bits following that message block, and tries to re-establish the relationship therebetween. If the relationship is satisfied, the message block is considered without error. Otherwise, an error has occurred during the transmission of that block. This method is further explained in greater detail below.

First, a CRC generating polynomial, $g_l(x)$, of order 1, is chosen. A common way of choosing the CRC generating polynomial is that $g_l(x)$ should satisfy $gcd(g_l(x), x) = 1$ for each and every i between 0 and 1, inclusive, wherein 1 and i are integers, and the function $gcd(A(X).B(x))$ is defined as the greatest common divider of polynomials A(X) and B(x). Examples of suitable g(x) include $g_4(x) = x^4 + x^3 + x^2 + x + 1$ for l=4; $g_7(x) = x^7 + x^6 + x^4 + 1$ for l=7:$g_8(x) = x^8 + x^7 + x^4 + x^3 + x + 1$ for l=8; and $g(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$ for l=12. The information of CRC generating polynomial is stored in both the transmitter and the receiver. **(Shien S.L, 2007)**

Shien, S.L., Industrial Technology Research Institute, 2007. *Cyclic redundancy check modification for message length detection and error detection*. U.S. Patent 7,240,273.

2D parity implemented in the previous lab is prone to leave errors made in a square pattern undetected, a large part of the burst errors can be detected using this method, although still another huge segment of the burst errors go undetected, this is where the CRC comes in, the number of bits in CRC is fixed, i.e. the number of redundant bits are fixed $O(1)$, unlike where in 2D parity the number of bits increase $O(n)$ as the message length increases, in terms of space complexity CRC is better.

Albeit there are a few limitations associated with CRC, such as:

(i) The error in the data can do undetected if the message is mutated such that it is a factor of the CRC polynomial itself.

(ii) Because a CRC is based on division, no polynomial can detect errors consisting of a string of zeroes prepended to the data, or of missing leading zeroes.

(iii) Single bit errors will be detected by any polynomial with at least two terms with non-zero coefficients.

(iv) Burst errors of length n will be detected by any polynomial of degree n or greater which has a non-zero x^0 term.

5. Conclusions

CRC is a much better upgrade to the 2D parity methods, with very low probability of errors left undetected by the receiver. It is sweet, short, although takes a little amount of computation, which can be fixed by using a lookup table, works great and is used practically in Networking.

6. Comments

a. Limitations of the experiment

The implementation part of the program is inefficient, i.e. it calculates the CRC for every byte in the data, although this is not required, since a lookup table can be created that will solve the problem in $O(1)$ time for every bit, and $O(n)$ in total, where n is the number of bytes in the message.

b. Limitations of the results obtained

The results are checked and the limitation of the CRC is shown, other than that and a few other limitations due to the maths behind CRC, no other limitations are known.

c. Learning

Through this lab the concept of CRC was learnt that is used for error detection.

d. Recommendations

Make the implementation more generic and more efficient, the program should work for any sort of data, files, strings, everything, this can be achieved by treating the data as a character array of bytes, then each byte can be processed individually.