**SCHOOL OF COMPUTER SCIENCE**

**UNIVERSITY OF PETROLEUM AND ENERGY STUDIES**

**DEHRADUN, UTTARAKHAND**



**DATA COMMUNICATION AND NETWORKS LAB**

**LABORATORY FILE**

**(2024-2025)**

**For**

**Vth Semester**

**Submitted To: Submitted By:**

Prof. Abhishek Yadav Mr. Akshat Negi

Assistant Professor S.S. 500106533(SAP ID)

[Vth Semester] R2142220414(Roll No.)

School of Computer Sciences B.Tech. CSF (Batch-1)

**LAB EXPERIMENT – 3**

**CRC and Hamming code**

A **CRC (Cyclic Redundancy Check) Experiment Lab** is a practical exercise to demonstrate error detection in digital communication. It involves implementing and testing CRC algorithms using a sender-receiver model. Here's how you can structure the lab:

**Objective**

1. Understand how CRC works for error detection.
2. Implement a CRC algorithm in C.
3. Simulate data transmission with errors and validate error detection.

**Requirements**

1. A C compiler (e.g., GCC).
2. A computer to run the code.

#include<stdio.h>

#include<string.h>

#define N strlen(gen\_poly)

char data[28];

char check\_value[28];

char gen\_poly[10];

int data\_length,i,j;

void XOR()

    {

        for(j = 1; j < N; j++)

        check\_value[j] = (( check\_value[j] == gen\_poly[j])?'0':'1');

    }

void receiver()

{

    printf("Enter the received data: ");

    scanf("%s", data);

    printf("Data received: %s", data);

    crc();

    for(i=0;(i<N-1) && (check\_value[i]!='1');i++);

        if(i<N-1)

            printf("\nError detected\n\n");

        else

            printf("\nNo error detected\n\n");

}

void crc()

{

    for(i=0;i<N;i++)

        check\_value[i]=data[i];

    do{

        if(check\_value[0]=='1')

            XOR();

        for(j=0;j<N-1;j++)

            check\_value[j]=check\_value[j+1];

        check\_value[j]=data[i++];

    }

    while(i<=data\_length+N-1);

}

int main()

{

    printf("\nEnter data to be transmitted: ");

    scanf("%s",data);

    printf("\nEnter the CRC polynomial: ");

    scanf("%s",gen\_poly);

    data\_length=strlen(data);

    for(i=data\_length;i<data\_length+N-1;i++)

        data[i]='0';

    printf("Data padded with n-1 zeros : %s",data);

    crc();

    for(i=data\_length;i<data\_length+N-1;i++)

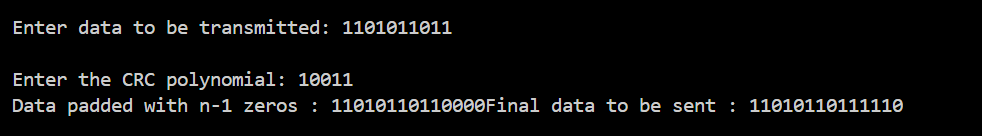
        data[i]=check\_value[i-data\_length];

    printf("Final data to be sent : %s\n\n",data);

    receiver();

        return 0;

}



A **Hamming Code Lab Experiment** focuses on implementing and testing error detection and correction using Hamming codes. This practical exercise demonstrates the ability to detect and correct single-bit errors in data transmission.

**Objective**

1. Understand the principles of Hamming codes for error detection and correction.
2. Implement Hamming code generation, error simulation, and error correction.
3. Experiment with single-bit error correction in transmitted data.

**Requirements**

1. A C compiler (e.g., GCC) or any programming environment.
2. Basic understanding of binary representation.

#include <string.h>

#include <math.h>

#include <stdlib.h>

#include <stdio.h>

    int MaxLength;

    int length;

    int parity;

    char \*HammingString=NULL;

    void EnterParameters(int \**length*, int \**parity*)

    {

        printf("Enter the maximum length: ");

        scanf("%d", *length*);

        printf("Enter the parity (0=even, 1=odd): ");

        scanf("%d", *parity*);

    }

    void CheckHamming(char \**HammingString*, int *parity*)

    {

        int i, j, k, start, length, ParityNumber;

        printf("Enter the Hamming code: ");

        scanf("%s", *HammingString*);

        int ErrorBit = 0;

        length = strlen(*HammingString*);

        length--;

        if (length > MaxLength)

        {

            printf("\n\*\* Invalid Entry - Exceeds Maximum Code Length of %d\n\n", MaxLength);

            return;

        }

        ParityNumber = ceil(log(length)/log(2));

        for(i = 0; i < ParityNumber; i++)

        {

            start = pow(2, i);

            int ParityCheck = *parity*;

            for(j = start; j < length; j=j+(2\*start))

            {

                for(k = j; (k < ((2\*j) - 1)) && (k < length); k++)

                {

                    ParityCheck ^= (*HammingString*[length - k] - '0');

                }

            }

                ErrorBit = ErrorBit + (ParityCheck \* start);

            }

        if(ErrorBit == 0)

        {

            printf("No error \n");

        }

        else

        {

            printf("There is an error in bit: %d\n", ErrorBit);

            if(*HammingString*[length - ErrorBit] == '0')

            {

*HammingString*[length - ErrorBit] = '1';

            }

            else

            {

*HammingString*[length - ErrorBit] = '0';

            }

            printf("The corrected Hamming code is: %s \n", *HammingString*);

        }

    }

    int main()

    {

        int parity;

        int choice = 0;

            printf("Error detection/correction: \n");

            printf("----------------------------\n");

            printf("1) Enter parameters \n");

            printf("2) Check Hamming code \n");

            printf("3) Exit \n");

            printf("\nEnter selection: ");

            scanf("%d", &choice);

            while (choice != 3)

            {

                if (choice == 1)

                {

                    EnterParameters(&MaxLength, &parity);

                    HammingString = (char\*) malloc (MaxLength \* sizeof(char));

                    main();

                }

                else if (choice == 2)

                {

                    CheckHamming(HammingString, parity);

                    main();

                }

                else

                {

                    printf("Valid options are 1, 2, or 3. Quitting program. \n");

                    exit(0);

                }

            }

            exit(0);

    }

