NAME : Vinit Dhoot  
CLASS : SY-IT[A]

ROLL NO. : 45

SUBJECT : Computer Networks  
ASSIGNMENT NO. : 03

Problem Statement : Implement hamming code single error correction method

CODE :

#include <stdio.h>

#include <Math.h>

#include <string.h>

char bits[4];

void calculateParityBit(char dataWithParity[], int parityPosition[])

{

    printf("\n\nCalculating Parity Bits.......\n");

    for (int i = 0; i < 4; i++)

    {

        printf("\nCalculating P%d...\n", parityPosition[i]+1);

        int count = 0, count1 = 0;

        int stepSize = parityPosition[i] + 1;

        for (int j = 10 - parityPosition[i]; j >= 0; j--)

        {

            printf("%c ", dataWithParity[j]);

            if (dataWithParity[j] == '1')

            {

                count++;

            }

            count1++;

            if (count1 == stepSize)

            {

                j = j - stepSize;

                count1 = 0;

            }

        }

        if (count % 2 == 0)

        {

            dataWithParity[10 - parityPosition[i]] = '0';

            printf("\n=> P%d = %d", parityPosition[i]+1, 0);

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

        }

        else

        {

            dataWithParity[10 - parityPosition[i]] = '1';

            printf("\n=> P%d = %d", parityPosition[i]+1, 1);

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

        }

    }

}

void calculateParityBitForReceiver(char finalData[], int parityPosition[])

{

    printf("\n\nCalculating Parity Bits.......\n");

    for (int i = 0; i < 4; i++)

    {

        printf("\nCalculating P%d...\n", parityPosition[i]+1);

        int count = 0, count1 = 0;

        int stepSize = parityPosition[i] + 1;

        for (int j = 10 - parityPosition[i]; j >= 0; j--)

        {

            printf("%c ", finalData[j]);

            if (finalData[j] == '1')

            {

                count++;

            }

            count1++;

            if (count1 == stepSize)

            {

                j = j - stepSize;

                count1 = 0;

            }

        }

        if (count % 2 == 0)

        {

            bits[i] = '0';

            printf("\n=>P%d = %d", parityPosition[i]+1, 0);

        }

        else

        {

            bits[i] = '1';

            printf("\n=>P%d = %d", parityPosition[i]+1, 1);

        }

    }

}

int binaryToDecimal(char \*binary) {

    int length = strlen(binary);

    int decimal = 0;

    for (int i = length - 1; i >= 0; i--) {

        if (binary[i] == '1') {

            decimal += 1 << (length - 1 - i);

        }

    }

    printf("\nError Position = %d ",decimal);

    printf("\n");

    return decimal;

}

int isPowerOfTwo(int num)

{

    return (num > 0) && ((num & (num - 1)) == 0);

}

void printData(char data[]){

    for(int i = 0 ; i<11;i++){

        printf("%c ",data[i]);

    }

}

void printBits(char bit[]){

    for(int i = 0 ; i<4;i++){

        printf("%c ",bit[i]);

    }

}

int main()

{

    char data[7];

    char dataWithParity[11];

    printf("\nEnter data :");

    scanf("%s", data);

    int parityPosition[4];

    int j = 0;

    for (int i = 10; i >= 0; i--)

    {

        if (!isPowerOfTwo(i + 1))

        {

            dataWithParity[10 - i] = data[j];

            j++;

        }

        else

        {

            dataWithParity[10 - i] = '\_';

        }

    }

    printf("\nData after appending parity positions : ");

    printData(dataWithParity);

    for (int i = 0; i < 4; i++)

    {

        parityPosition[i] = (int)pow(2, i) - 1;

    }

    // calculating pariy bits

    calculateParityBit(dataWithParity, parityPosition);

    printf("\nData after setting parity bits: ");

    printData(dataWithParity);

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

    // store datawithParity in finalData

    char finalData[11];

    for (int i = 0; i < 11; i++)

    {

        finalData[i] = dataWithParity[i];

    }

    printf("\nMenu : \n1)Send data with error\n2)Send data without Error\nEnter choice : ");

    int ch;

    int errorPos;

    scanf("%d", &ch);

    switch (ch)

    {

    case 1:

        printf("\nEnter error position [1 to 11] : ");

        scanf("%d", &errorPos);

        if (finalData[11 - errorPos] == '1')

        {

            finalData[11 - errorPos] = '0';

        }

        else

        {

            finalData[11 - errorPos] = '1';

        }

        printf("\nFinal data after introducing error: ");

        printData(finalData);

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

        // calculate parity bits

        calculateParityBitForReceiver(finalData, parityPosition);

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

        printf("\nCalculated Parity Bits (Receiver side) : ");

        printBits(bits);

        printf("\n");

        printf("\nReversing Parity bits array : ");

        int s1 = 0;

        int e1 = 3;

        while (s1 < e1)

        {

            char temp = bits[s1];

            bits[s1] = bits[e1];

            bits[e1] = temp;

            s1++;

            e1--;

        }

        printBits(bits);

        printf("\n");

        // finding error position

        int pos = binaryToDecimal(bits);

        printf("\nFinal Data afrer Fixing error => ");

        if (finalData[11-pos] == '1')

        {

            finalData[11-pos] = '0';

        }

        else

        {

            finalData[11-pos] = '1';

        }

        printData(finalData);

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

        break;

    case 2:

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

        // calculate parity bits

        calculateParityBitForReceiver(finalData, parityPosition);

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

        printf("\nCalculated Parity Bits (Receiver side) : ");

        for (int u = 0; u < 4; u++)

        {

            printf("%c ", bits[u]);

        }

        printf("\n");

        printf("\nNo Error Detected.");

        printf("\n");

        printf("\nFinal Data => ");

        for (int t = 0; t < 11; t++)

        {

            printf("%c ", finalData[t]);

        }

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

        break;

    default:

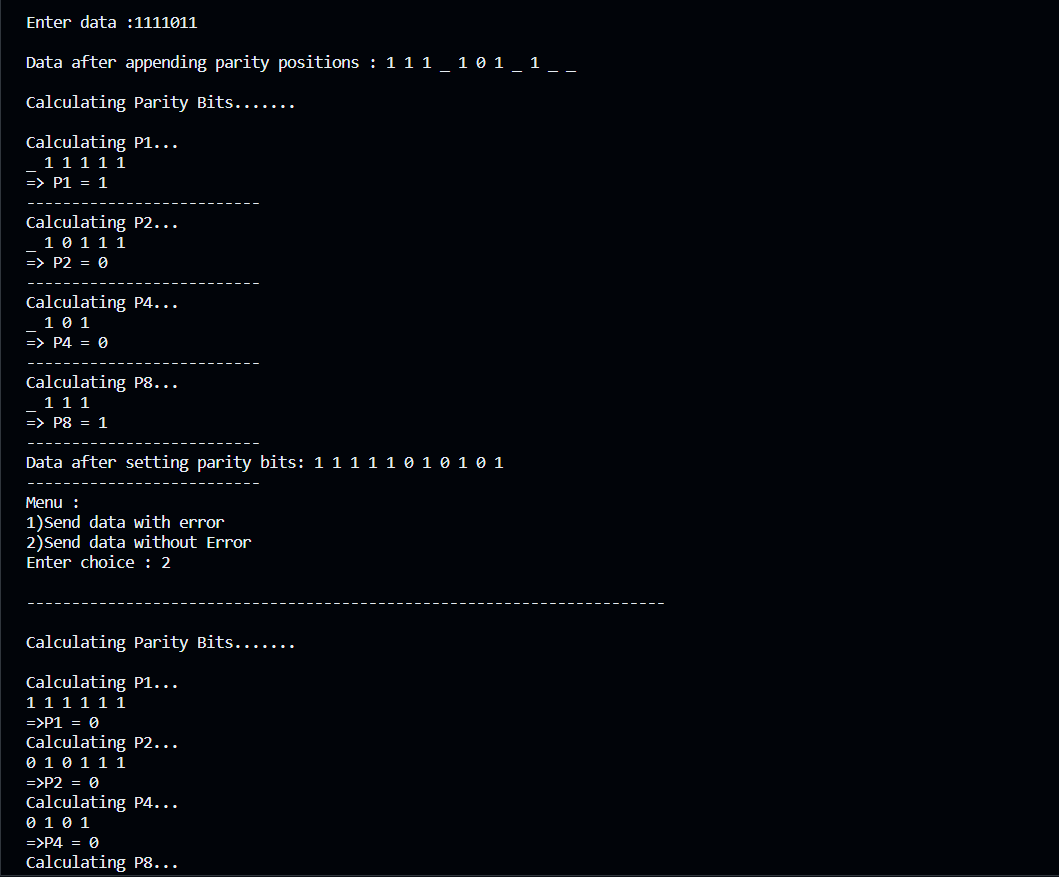
        break;

    }

}

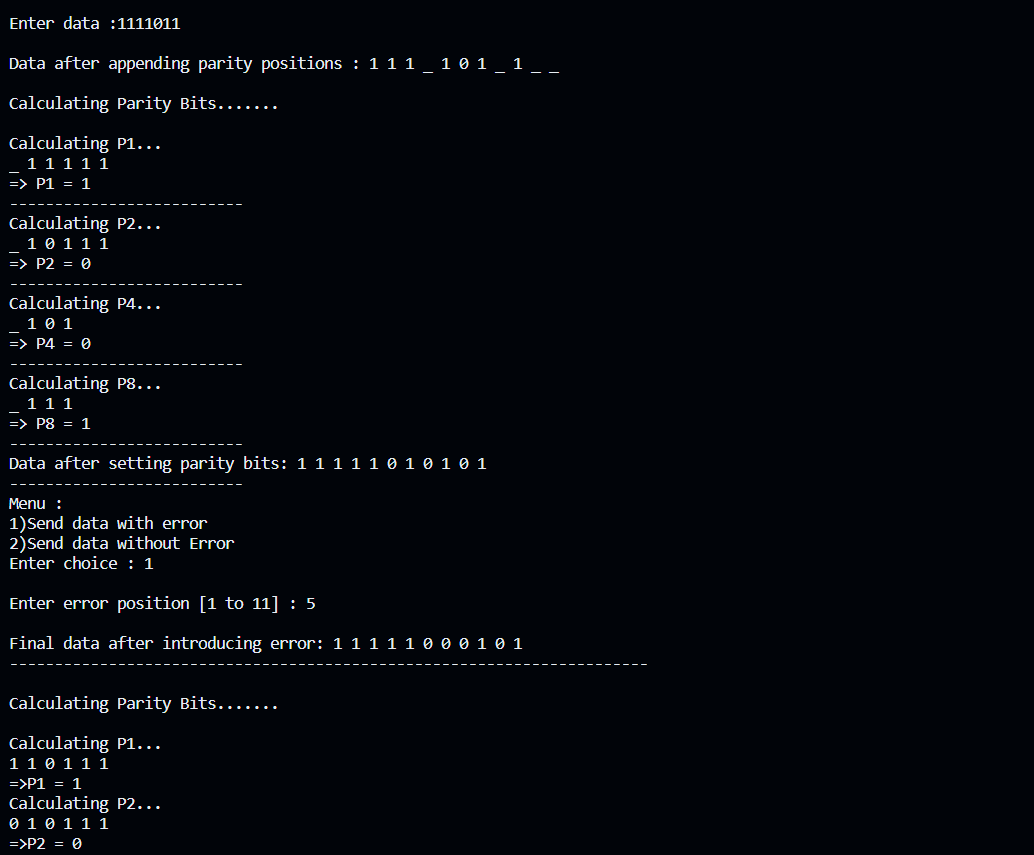
**Output:-**

**Without Error:-**

**A black screen with white dots

Description automatically generated**

**With Error:-**

A black background with white lines and numbers

Description automatically generated