Exc No: 06

Date : 7.8.24

Aim:

Write a program to implement error detection and correction using Hamming code concept. Make a test run to input data stream and verify error correction failure.

Error Correction at Data Link Layer:

Hamming code is a set of error - correction codes that can be used to detect and correct the errors that can occur when data is transmitted from the sender to the receiver.

Create Sender Program with below features:

1. Input to sender file should be test of any length: Program should convert the test of to binary.
2. Apply the hamming code concept on the binary data and add redundant bits to it

Create a receiver program with below features

1. Receiver program should read the input from the channel file.
2. Apply hamming code on the binary data to check for errors.
3. If there is an error, display the position of the error.

4. Else remove the redundant bits and convert the binary data to Ascii and display the output.

Student Observation:

Code:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
void charToBinary(char ch, int binary[], int *index) {
    for (int i = 7; i >= 0; i--) {
        binary[(*index)++] = (ch >> i) & 1;
    }
}
void calculateParityBits(int hammingCode[], int n, int r) {
    for (int i = 0; i < r; i++) {
        int parityPos = (int) pow(2, i);
        int parity = 0;
        for (int j = parityPos; j <= n; j += 2 * parityPos) {
            for (int k = j; k < j + parityPos && k <= n; k++) {
                parity ^= hammingCode[k];
```

```
        { parity ^ = hammingCode[k];
        }
    }
int generateHammingCode(int dataBit[], int m,
int hammingCode[]) {
        int r = 0;
        int h = m;
    while(m++r > pow(2,r)) {
            r++;
    }
    n = m+r;
    for(int i=1, j=0, k=0; i<=n; i++) {
        if(i == (int)pow(2,k)) {
            hammingCode[i] = 0;
            k++;
        }
        else { hammingCode[i] = dataBits[j++];
        }
    }
    calculateParityBits(hammingCode, n, r);
    return n;
}
int detectAndCorrectError(int hammingCode[], int n,
int r) {
        int errorPos = 0;
    for(int i=0; i<r; i++) {
            int parityPos = (int) pow(2,i);
            int parity = 0;
```

```
for ( int j= paritypos;

{ for (int k=j ; k<j + paritypos && k<=n ; k++)

        parity ^= hammingCode [k];

    }}
    if (parity != 0) {
        errorpos = paritypos;
    }
} }
return errorpos;
}
void binaryToChar (int binary[], int length,
char output[]) {

    int index = 0;

for ( int i= 0; i< length ; i+= 8) {

    char ch = 0;
  for ( int j=0 ; j< 8 ; j++ ) {
        ch |= (binary [i+j] << (7-j));
}
    output [index++] = '\0';

}                          // null.
int main () {
    char inputString [32];
    int binary [256];
    int dataBits [256];
```

```
int hammingCode[512];
printf("Enter input string: ");
scanf("%s", inputString);
int index = 0;
for (int i = 0; i < strlen(inputString); i++){
charToBinary(inputString[i], binary, &index);
}
for (int i=0; i<index; i++) {
    dataBits[i] = binary[i];
}
int n = generateHammingCode(dataBits, index,
hammingCode);
printf("Generated Hamming Code");
for (int i=1; i<=n; i++) {
    printf("%d", hammingCode[i]);
}
printf("\n");
int correctedDatabits[256];
int j=0, k=0;
for (int i=1; i<=n; i++) {
    if (i != (int) pow(2, k)) {
        correctedDatabits[j++] = hammingCode[i];
} else {
    k++;
}
}
```

```
    char corrected string[32];
    binaryTo char (correctedDataBits, i, corrected string);
    printf ("Corrected string : %s \n",
            correctedstring);
    return 0;
}
```

Input & output:

Enter the input string: hello

Generated Hamming Code : 1101 1110 1100 0001 1101
0101 0110 1100 1110 0011.

Enter the position to simulate error (0 for no error) : 3

Hamming Code with error : 1111 1110 0110 0001 1110
0101 0111 0011 0001 1110 0000 1010 0110 1110

Error detected at position : 3

Corrected hamming code: 1101 1110 1100 0001 1110
0101 0110 0111 0001 1101 0001 1001 0011 1111.

Corrected bit at position 3 : 0

Corrected string : hello.

Result: Thus the hamming code detection and correction is applied and verified.