Exp No: 6

Date: 21/8/24

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 feature.

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 the data is transmitted from the sender to the receiver. It is a technique developed by R.W. Hamming for error correction.

Create sender program with below features.

1. Input to sender file should be a text of any length. Program should convert the text to binary.
2. Apply hamming code concept on the binary data and add to check for errors.
3. Save this output in a file called channel.

Create a receiver program with below features

1. Receiver program should read the input from chanel 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:

```
#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]; } }
    hammingCode[parityPos] = parity;
    }
}
int generateHammingCode (int dataBits[], int m, int
                                hammingCode[]){
    int r=0;
    int n=m;
while (n+r+1 > 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; j <= n; j += (2 * 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++] = ch;
        output[index] = '\0';
    }
```

```
int main() {
    char inputString[32];
    int binary[256];
    int dataBits[256];
    int hammingCode[512];

    printf("Enter the input string: ");
    scanf("%s", inputString);

    for 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 = generatedHammingCode(dataBits, index,
                             hammingCode);
    printf("Generated Hamming Code: ");
    for (int i = 1; i <= n; i++) {
        printf("%d", hammingCode[i]);
    }
    printf("\n");

printf("Enter Position to simulate error: ");
    int errorPos;
    scanf("%d", &errorPos);

if (errorPos > 0 && errorPos <= n) {
    hammingCode[errorPos] = !hammingCode[errorPos];
    printf("Hamming code with error: ");
    for (int i = 1; i <= n; i++) {
        printf("%d", hammingCode[i]);
    }
    printf("\n");
}
```

```
int detectError = detectAndCorrectError(hammingCode,
                    n, log2(n+1));
if (detectedErrorPos == 0) {
    printf("No error detected.\n");
} else {
    printf("Error detected at position: %d\n",
                    detectedErrorPos);
int originalBit = !hammingCode[detected
                    ErrorPos];
hammingCode[detectedError] = originalBit;
printf("Corrected Hamming code:");

for (int i=1; i<=n; i++) {
    printf("%d ", hammingCode[i]);
}
printf("\n");
printf("Corrected Bit at position %d:",
                    detectedError, originalBit);
}

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++;
    }
}
binaryToChar(correctedDataBits, j,
                    correctedString);
printf("Corrected string: %s\n", correctedString);
return 0;
}
```

O/P:-

```
Enter the input string: ant
Generated Hamming code:
0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 0
1 0 0
Enter position to simulate error: 6

Hamming code with error:
0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 0
1 0 0

Error detected at position: 6
0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0 0 1 1 1 0
1 0 0

Corrected bit at position 6: 1
Corrected: ant
```

Result:

Thus the error detection & correction program using hamming code is successfully executed and output is verified.

24/8/21