Exp 6

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

Hamming code is a set of error-correction code 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.

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<n; 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++];
        }
    }
}
```

```
calculateParityBit(hammingCode, n, r);
    return n;
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 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 errorPos;
scanf("%d", &errorPos);
if (errorPos >0 && errorPos <=n){
    hammingcode[errorPos] = !hammingcode
                            [errorpos];
int detectedErrorPos = 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[detectedErrorPos];
hammingCode[detectedErrorPos] = originalBit;
```

```
printf("Corrected Hamming Code: ");
for(int i=1; i<=n; i++){
    printf("%d ", hammingCode[i]);
}
printf("\n");
printf("Corrected bit at position %d
    \n%d\n", detectedErrorPos,
    originalBit);}
int correctedDataBits[256];
int i=0, k=0;
for(int i=1, i<=n; i++){
    if(i != (int) pow(2,k)){
        correctedDataBits[j++]
    char correctedString[32];
    binaryToChar(correctedDataBits, correctedString);
printf("Corrected String: %s\n",
    correctedString);
return 0;
}
```

Input & Output:

enter the input string: hello

Generated Hamming Code: 11011101100001110

01010110011000111011000110111

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

Hamming code with error: 11111101100001110

01010110011000111011000110111

Error detected at Position: 3

Corrected bit at position 3: 0

corrected string: hello

Result:

Thus the hamming code detection and correction is applied and verified the output.

18/9/24