16/08/2024

Practical - 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.

Code (binary)

```
#include <stdio.h>

void calculate_parity_bits(int data[], int * p1, int *p2,
                           int *p4, int *p8){
    *p1 = (data[0] + data[2] + data[3] + data[5] + data[6])%2;
    *p2 = (data[0] + data[1] + data[3] + data[4] + data[6])%2;
    ~~*p3~~
    *p4 = (data[3] + data[4] + data[5])%2;
    *p8 = (data[1] + data[0] + data[2])%2;
}

void parity_bits(int data[], int *p1, int *p2, int *p4,
                 int *p8){
    *p1 = (data[10] + data[8] + data[6] + data[4] + data[2]
           + data[0])%2;
    *p2 = (data[9] + data[8] + data[5] + data[4] + data[1]
           + data[0])%2;
    *p4 = (data[7] + data[6] + data[5] + data[4])%2;
    *p8 = (data[3] + data[2] + data[1])%2;
}

void generate_hamming_code(int data[], int hamming_code[]){
    int p1, p2, p4, p8;
    calculate_parity_bits(data, &p1, &p2, &p4, &p8);
    hamming_code[0] = data[0];
    hamming_code[1] = data[1];
    hamming_code[2] = data[2];
    hamming_code[3] = p8;
    hamming_code[4] = data[3];
    hamming_code[5] = data[4];
    hamming_code[6] = data[5];
    hamming_code[7] = ~~data~~ p4;
    hamming_code[8] = data[6];
    hamming_code[9] = p2;
    hamming_code[10] = p1;
}
```

```
int detect_error (int hamming_code [ ]){
    int p1, p2, p4, p8;
    parity_bits (hamming_code, &p1, &p2, &p4, &p8);
    int error_position = p1*1 + p2*2 + p4*4 + p8*8;
    return error_position;
}

int main (){
    int data [7];
    printf ("Enter 7 bits of data one by one\n");
    for (int i=0; i<7; i++) {
        printf ("Bit %d: ", i+1);
        scanf ("%d", &data [i]);
    }
    printf ("Data after appending all bits: ");
    for (int i = 0; i<7; i++){
        printf ("%d", data [i]);
    }
    printf ("\n");
    int hamming_code [11];
    generate_hamming_code (data, hamming_code);
    printf ("The 11-bit hamming code is: ");
    for (int i=0; i<11; i++) {
        printf ("%d", hamming_code [i]);
    }
    printf ("\n");
    int corrupted_code [11];
    printf ("Enter the 11-bit Hamming code with
        a possible error (bit by bit): \n");
    for (int i =0; i<11; i++) {
        printf ("Bit %d: ", i+1);
        scanf ("%d", &corrupted_code [i]);
}
```

```
int error_pos = detect_error(corrupted_code);
printf("Calculated error position : %d\n",
                                  11-error_pos+1);
if(corrupted_code[11-error_pos]==0){
    corrupted_code[11-error_pos]=1;
}else{
    corrupted_code[11-error_pos]=0;
}

printf("Data after error-correcting all bits:");
for(int i=0; i<11; i++){
    printf("%d", corrupted_code[i]);
}
printf("\n");
return 0;
}
```

Output

Enter 7 bits of data one by one:

Bit 1:1

Bit 2:1

Bit 3:0

Bit 4:0

Bit 5:1

Bit 6:1

Bit 7:1

Data after appending all bits : 1100111

The 11-bit Hamming code is : 11000110101

Enter the 11-bit hamming code with a possible error (bit by bit):

Bit 1:1

Bit 2:1

Bit 3:1

Bit 4:0

Bit 5:0

Bit 6:1

Bit 7: 1

Bit 8: 0

Bit 9: 1

Bit 10: 0

Bit 11: 1

Calculated error position: 3

Data after error-correcting all bits: 11000110101

RESULT

Thus, a program ~~to~~ to be implement error detection and correction using hamming code concept has been successfully written and the output is verified for binary input.

2/9/19

Code (Text)

```
#include <stdio.h>
#include <string.h>

void calculate_parity_bits (int data[], int *p1,
    int *p2, int *p4, int *p8) {
    *p1 = (data[0] + data[2] + data[3] + data[5] +
                                    data[6]) % 2;
    *p2 = (data[0] + data[1] + data[3] + data[4] +
                                    data[6]) % 2;
    *p4 = (data[2] + data[4] + data[5]) % 2;
    *p8 = (data[1] + data[0] + data[2]) % 2;
}

void parity_bits (int data[], int *p1, int *p2,
                                int *p4, int *p8) {
    *p1 = (data[10] + data[8] + data[6] + data[4] +
                                data[2] + data[0]) % 2;
    *p2 = (data[9] + data[8] + data[5] + data[4] +
                                data[1] + data[0]) % 2;
    *p4 = (data[7] + data[6] + data[5] + data[4]) % 2;
    *p8 = (data[3] + data[3] + data[2] + data[1]) % 2;
}

void generate_hamming_code (int data[], int
                                hamming_code[]) {
    int p1, p2, p4, p8;
    calculate_parity_bits (data, &p1, &p2, &p4, &p8);
    hamming_code[0] = data[0];
    hamming_code[1] = data[1];
    hamming_code[2] = data[2];
    hamming_code[3] = p8;
    hamming_code[4] = data[3];
    hamming_code[5] = data[4];
    hamming_code[6] = data[5];
    hamming_code[7] = p4;
    hamming_code[8] = data[6];
    hamming_code[9] = p2;
    hamming_code[10] = p1;
}
```

```
int detect_error (int hamming_code [])
    int p1, p2, p4, p8;
    parity_bits (hamming_code, &p1, &p2, &p4, &p8);
    int error_position = p1 * 1 + p2 * 2 + p4 * 4 + p8 * 8;
    return error_position;
}

void char_to_binary (char ch, int data[]) {
    for (int i = 0; i < 7; i++) {
        data [i] = (ch >> (6-i)) & 1;
    }
}

char binary_to_char (int data [])
    char ch = 0;
    for (int i = 0; i < 7; i++) {
        ch |= (data [i] << (6-i));
    }
    return ch;
}

int main () {
    char input_text [100];
    printf ("Enter a string (up to 99 characters): \n");
    scanf ("%s", input_text);
    int data [7], hamming_code [11], corrupted_code [11];
    char encoded_text [1100] = {0};
    char decoded_text [100] = {0};
    int k = 0;

    for (int i = 0; i < strlen (input_text); i++) {
        char_to_binary (input_text [i], data);
        generate_hamming_code (data, hamming_code);
        for (int j = 0; j < 11; j++) {
            encoded_text [k++] = hamming_code [j] + '0';
        }
    }
```

```
printf("Encoded hamming code: %s\n", encoded_text);

printf("Enter the received Hamming code (up to
        1100 bits, same length as encoded):\n");
scanf("%s", encoded_text);

k = 0;

for (int i = 0; i < strlen(encoded_text)/11; i++){
    for (int j = 0; j < 11; j++) {
        corrupted_code[j] = encoded_text[k++]
                                    - '0';
    }

    int error_pos = detect_error(corrupted_code);
    if(error_pos != 0)
        corrupted_code[11 - error_pos] =
            (corrupted_code[11 - error_pos] == 0) ?
                                                1 : 0;
    }

    data[0] = corrupted_code[0];
    data[1] = corrupted_code[1];
    data[2] = corrupted_code[2];
    data[3] = corrupted_code[3];
    data[4] = corrupted_code[4];
    data[5] = corrupted_code[5];
    data[6] = corrupted_code[6];

    decoded_text[i] = binary_to_char(data);
}

printf("Decoded and corrected text: %s\n",
                                decoded_text);

return 0;

}
```

OUTPUT

Enter a string (up to 99 characters):

abc

Encoded Hamming Code: 110000001101000011000110000011111

Enter the received Hamming Code (up to 11500 bits, same length as encoded):

110000001101000011000110000011110

Decoded and corrected text: abc

RESULT

Thus, a program to implement ~~flow control~~ error detection and correction using hamming code concept has been written and successfully verified for test inputs.

27/9/24