EX9 – HAMMING CODE

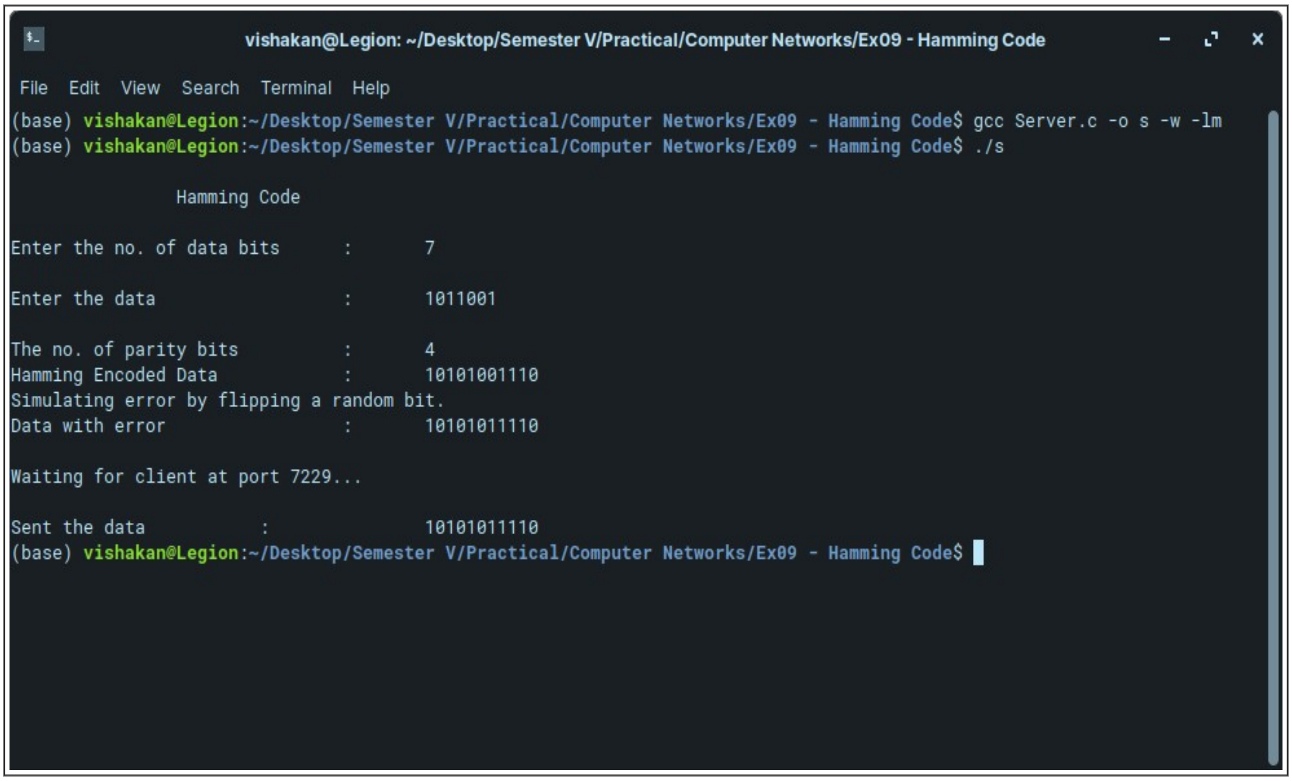
- S. Vishakan CSE – C 18 5001 196

Server Program:

```
#include "Hamming.h"
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 7229

int main(void){
        int sockfd, newfd, len, flag, i, *hammed_data, *error_data;
        int data_bits, total_bits, parity_bits;
        struct sockaddr_in server_address, client_address;
        char buffer[1024];

        printf("\n\t\tHamming Code\n");
        printf("\nEnter the no. of data bits\t:\t");
        scanf("%d", &data_bits);

        printf("\nEnter the data\t\t\t:\t");

        int data[data_bits];

        for(i = 0; i < data_bits; i++){
                scanf("%1d", &data[i]);
        }

parity_bits = findParityBits(data_bits);
printf("\nThe no. of parity bits\t\t:\t%d", parity_bits);
total_bits = parity_bits + data_bits;

hammed_data = putParityBits(data, data_bits);
printf("\nHamming Encoded Data\t\t:\t");
printMessage(hammed_data, total_bits);

printf("\nSimulating error by flipping a random bit.");
error_data = flipABit(hammed_data, total_bits);

printf("\nData with error\t\t\t:\t");
printMessage(error_data, total_bits);

for(i = 0; i < total_bits; i++){
buffer[i] = error_data[i] + '0';
}
```

```
    buffer[i] = '\0';

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if(sockfd < 0){ //Error has occurred.
        perror("Socket cannot be created.\n");
        exit(1);
    }

    bzero(&server_address, sizeof(server_address));

    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);

    if(bind(sockfd, (struct sockaddr*)&server_address, sizeof(server_address)) < 0){
        perror("Bind error occurred.\n");
        exit(1);
    }

    printf("\n\nWaiting for client at port %d...\n", PORT);
    listen(sockfd, 2);
    len = sizeof(client_address);

    newfd = accept(sockfd, (struct sockaddr*)&client_address, &len);

    flag = send(newfd, buffer, sizeof(buffer), 0);
    printf("\nSent the data\t\t:\t\t%s\n", buffer);

    close(sockfd);
    close(newfd);

    return 0;
}
```

Output:



vishakan@Legion: ~/Desktop/Semester V/Practical/Computer Networks/Ex09 - Hamming Code
File Edit View Search Terminal Help

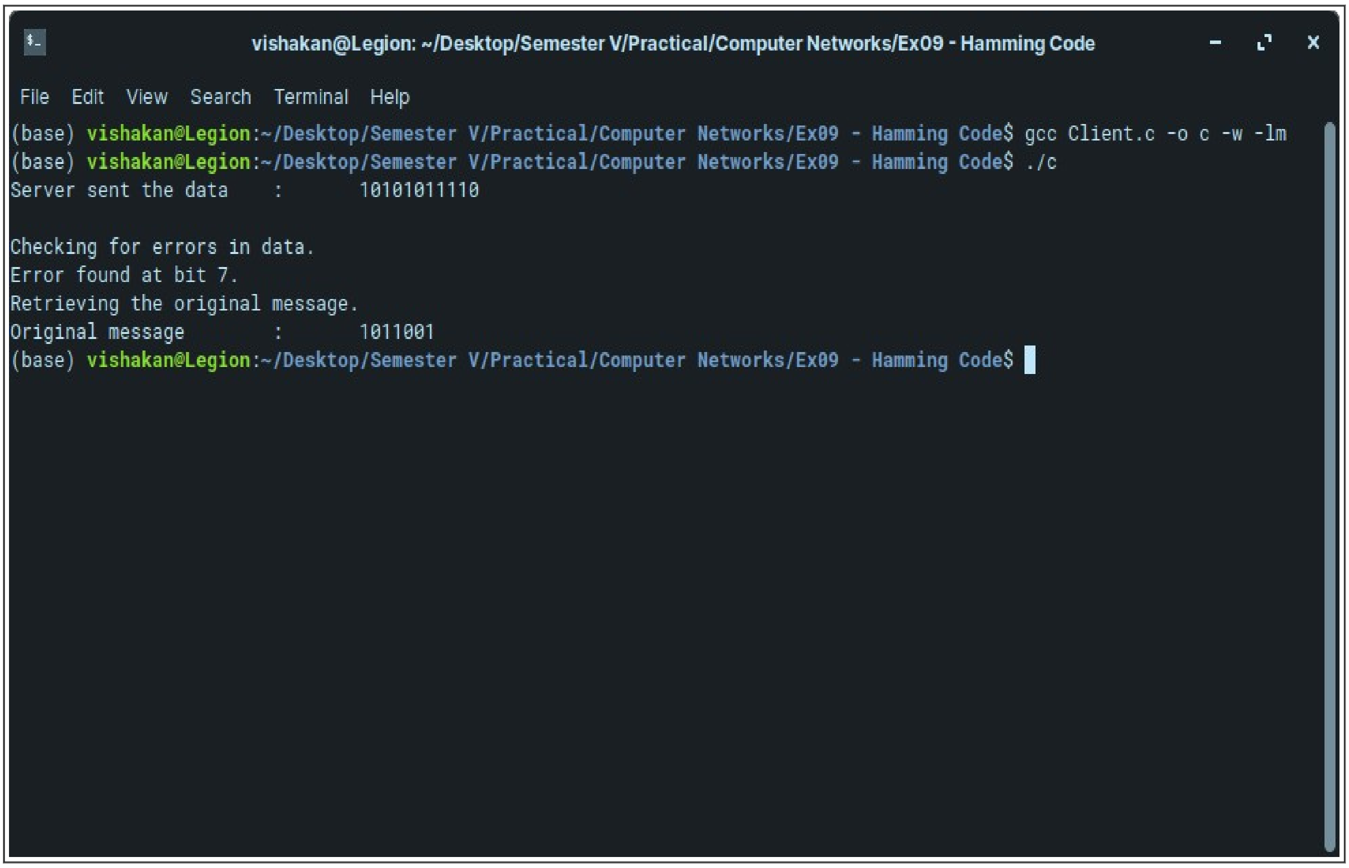
```
(base) vishakan@Legion:~/Desktop/Semester V/Practical/Computer Networks/Ex09 - Hamming Code$ gcc Server.c -o s -w -lm
(base) vishakan@Legion:~/Desktop/Semester V/Practical/Computer Networks/Ex09 - Hamming Code$ ./s

                Hamming Code

Enter the no. of data bits      :       7

Enter the data                  :       1011001

The no. of parity bits          :       4
Hamming Encoded Data            :       10101001110
Simulating error by flipping a random bit.
Data with error                 :       10101011110

Waiting for client at port 7229...

Sent the data           :               10101011110
(base) vishakan@Legion:~/Desktop/Semester V/Practical/Computer Networks/Ex09 - Hamming Code$ 
```

Client Program:

```
#include "Hamming.h"
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define PORT 7229

int main(void){
	int sockfd, flag, len, i;
	int *data, data_bits, parity_bits, total_bits;
	struct sockaddr_in server_address, client_address;
	char buffer[1024];

	sockfd = socket(AF_INET, SOCK_STREAM, 0);

	if(sockfd < 0){
		perror("Socket cannot be created.\n");
		exit(1);
	}

	bzero(&server_address, sizeof(server_address));

	server_address.sin_family = AF_INET;
	server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
	server_address.sin_port = htons(7229);

	connect(sockfd, (struct sockaddr*)&server_address, sizeof(server_address));

	flag = recv(sockfd, buffer, sizeof(buffer), 0);

	printf("Server sent the data\t:\t%s\n", buffer);

	total_bits = strlen(buffer);

	int hammed_data[total_bits];

	for(i = 0; i < total_bits; i++){
		hammed_data[i] = buffer[i] - '0';
	}

	parity_bits = findParityBits(total_bits);

	printf("\nChecking for errors in data.");
	data = detectError(hammed_data, total_bits, parity_bits);

	printf("\nRetrieving the original message.");
	data = getMessage(data, total_bits, parity_bits);
```

```c
    printf("\nOriginal message\t:\t");
    printMessage(data, total_bits – parity_bits);
    printf("\n");


    close(sockfd);


    return 0;
}
```

Output:


vishakan@Legion: ~/Desktop/Semester V/Practical/Computer Networks/Ex09 - Hamming Code
File
Edit
View
Search
Terminal
Help

```
(base) vishakan@Legion:~/Desktop/Semester V/Practical/Computer Networks/Ex09 - Hamming Code$ gcc Client.c -o c -w -lm
(base) vishakan@Legion:~/Desktop/Semester V/Practical/Computer Networks/Ex09 - Hamming Code$ ./c
Server sent the data    :       10101011110

Checking for errors in data.
Error found at bit 7.
Retrieving the original message.
Original message        :       1011001
(base) vishakan@Legion:~/Desktop/Semester V/Practical/Computer Networks/Ex09 - Hamming Code$ 
```

Header File “Hamming.h”:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int findParityBits(int msg_bits);
int *putParityBits(int *data, int data_bits);
int *flipABit(int *data, int bits);
int *detectError(int *array, int bits, int parity_bits);
int *getMessage(int *hammed_data, int bits, int parity_bits);
void printMessage(int *data, int bits);
long convertBinToDec(long bin_value);
int *reverseArray(int *array, int len);



int findParityBits(int msg_bits){
        //Calculates the number of parity bits required
        int i = 0;

        //No. of redundant bits are always power of 2 that is greater than the whole
        message size
        while(pow(2, i) < msg_bits + i + 1){
                i += 1;
        }

        return i;
}

int *putParityBits(int *data, int data_bits){
        //Places the required parity bits on a given array of data
        //Even parity is assumed here
        int i = 0, j = 0, k = 0, parity_bits = 0, total_bits = 0;
        int *hammed_data, bit_value = 0, parity_pos = 0;

        parity_bits = findParityBits(data_bits);
        total_bits = parity_bits + data_bits;
        hammed_data = (int *)malloc(sizeof(int) * total_bits);

        data = reverseArray(data, data_bits);

        for(i = 0; i < total_bits; i++){
                //Insert empty parity bits at the correct position
                if(i == ((1 << j) - 1)){ // 1 << i == 2^i in binary
                        hammed_data[i] = 0;
                        j++;
                }
```

```
		else{
			hammed_data[i] = data[k];
			k++;
		}
	}

	for(i = 0; i < total_bits; i++){
		//Calculate the parity values for each parity bit

		for(j = 0; j < parity_bits; j++){
			bit_value = ((i + 1) & (1 << j));

			if(bit_value){
			//Implies the index value is not a power of 2, thus a data bit is at 'i'
				parity_pos = (1 << j) - 1;
				hammed_data[parity_pos] = hammed_data[parity_pos] ^
				hammed_data[i];
			}
		}
	}

	return reverseArray(hammed_data, total_bits);
}

int *flipABit(int *data, int bits){
	//Flips a random bit in the data to simulate error
	int pos;

	pos = rand() % bits;
	data[pos] = (data[pos] + 1) % 2;

	return data;
}

int *detectError(int *data, int bits, int parity_bits){
	//Detects the error (and corrects, if possible)
	int i = 0, j = 0, parity_pos, bit_value;
	int new_parity = 0, error_pos = 0, *new_data;

	data = reverseArray(data, bits);

	for(j = parity_bits - 1; j >= 0; j--){
		//Calculate the parity values again for each parity bit
		//and compare

		new_parity = 0;

		for(i = 0; i < bits; i++){
			bit_value = ((i + 1) & (1 << j));
```

```
            if(bit_value){
        //Implies the index value is not a power of 2, thus a data bit is at 'i'
                new_parity = new_parity ^ data[i];
            }
        }

        error_pos += new_parity * (1 << j);
    }

    if(error_pos){
        printf("\nError found at bit %d.", bits - error_pos + 1);
        data[error_pos - 1] = (data[error_pos - 1] + 1) % 2;
    }

    else{
        printf("\nNo Error was found.");
    }

    return data;
}

int *getMessage(int *hammed_data, int bits, int parity_bits){
    //Retrieves the original message from the Hamming encoded data
    int *data, i = 0, j = 0, k = 0;

    data = (int *)malloc(sizeof(int) * (bits - parity_bits));

    for(i = 0; i < bits; i++){
        if(i != ((1 << j) - 1)){
            data[k++] = hammed_data[i];
        }
        else{
            j++;
        }
    }

    return reverseArray(data, bits - parity_bits);
}

void printMessage(int *data, int bits){
    //Prints a given array of data
    int i = 0;

    for(i = 0; i < bits; i++){
        printf("%d", data[i]);
    }

    return;
}
```

```
long convertBinToDec(long bin_value){
    //Converts a given binary value to decimal
    int dec_value = 0, i = 0, rem = 0;

    while(bin_value != 0){
        rem = bin_value % 10;
        bin_value /= 10;
        dec_value += rem * pow(2, i);
        i++;
    }

    return dec_value;
}

int *reverseArray(int *array, int len){
    //Reverses the given array of data
    int *rev, i = 0;
    rev = (int *)malloc(sizeof(int) * len);

    for(i = 0; i < len; i++){
        rev[i] = array[(len - 1) - i];
    }

    return rev;
}
```