Computer Networks

Lab 6

Jerin Jim 2022BCD0047

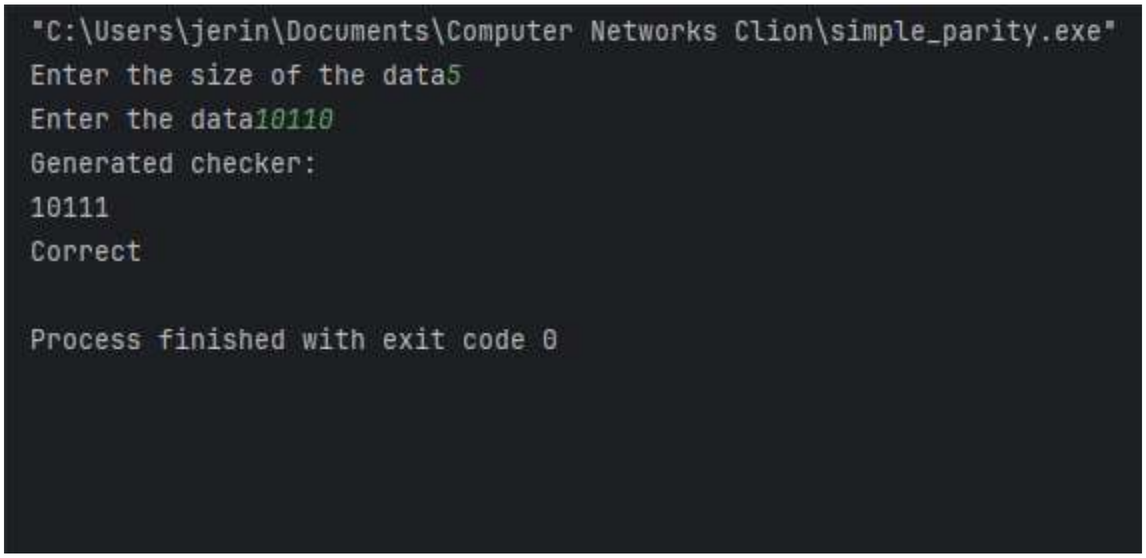
Question 1:

Write the C/C++ Program to implement simple parity check code to detect single bit error during the message transmission from sender to receiver.

Code:

```
#include <stdio.h>

int main()
{
    printf("Enter the size of the data");
    int len;
    scanf("%d", &len);
    printf("Enter the data");
    char a[100];
    for(int i = 0; i < len; ++i)//Input from the user (Sender side
Data)
    {
        scanf("%c", &a[i]);
    }
    int one,zero = 0;
    for(int i = 0; a[i] !='\0'; i++)//For loop to check the number of
1s and 0s in the data
    {
        if (a[i] == '1')
        {
            one+=1;
        }
        else
        {
            zero+=1;
        }
    }

    char add;
    if (one%2==0)//Add variable that determines if the parity bit to be
added is 1 or 0. 1 if the number of 1's are odd else 0
    {
        add = '0';
    }
    else
    {
        add = '1';
    }

    char chckr[100];//Final data
    int j =0;
    while (a[j] != '\0')
    {
        chckr[j] = a[j];
        j++;
    }

    chckr[j] = add;//Adding parity bit to the final data
    printf("Generated checker: %s\n",chckr);

    one = 0;
    for(int i = 0; chckr[i] !='\0'; i++)//For loop to check the data
from the receiver side
    {
        if (chckr[i] == '1')//Counts the number of 1's in the receiver
side
        {
            one+=1;
        }
    }
    if (one%2==0)//Correct if the number of 1's are even, else wrong
    {
        printf("Correct\n");
    }

    else
    {
        printf("Wrong\n");
    }
    return 0;
}
```

Output:



```
"C:\Users\jerin\Documents\Computer Networks Clion\simple_parity.exe"
Enter the size of the data5
Enter the data10110
Generated checker:
10111
Correct

Process finished with exit code 0
```

Working:

- The input is taken from the user, and the number of 1’s in the input is counted using a for loop and stored in the variable ‘one’.
- It is check if the value of ‘one’ is even or odd. That is, it checks if the cardinality of 1’s in the input is even or odd.
- The input is then copied to a new array chckr[]. It then adds 1 or 0 to the end of the chckr[] depending on if the value of ‘one’ is odd or even. If it is odd, then 1 is added to the end of the string, else 0 is added to the end.
- The data is verified on the receiver’s end by checking if the number of 1’s in the string is odd or even. If it is even, then the data is safe, else the data has been corrupted.

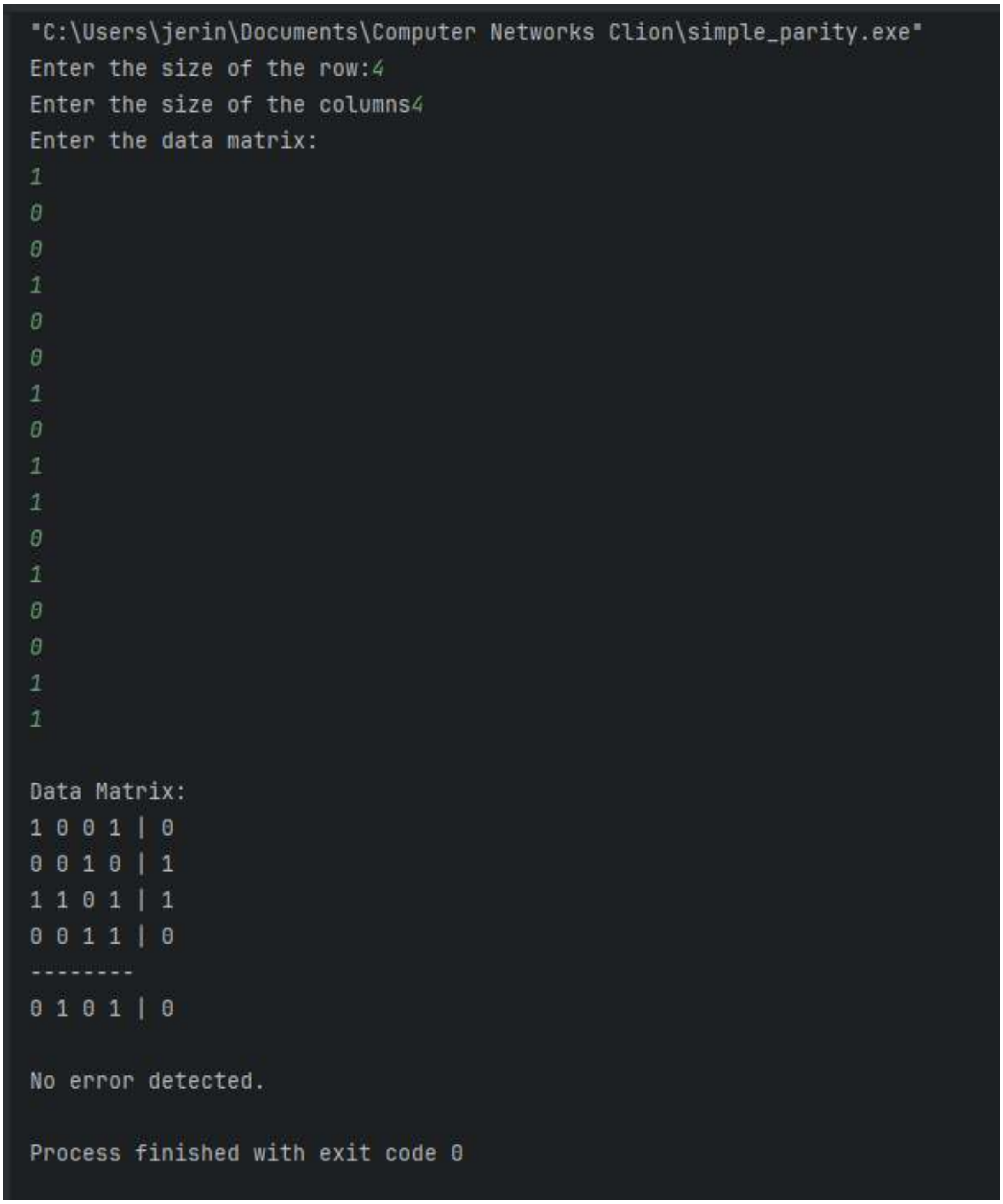
Question 2:

Write the C/C++ Program to implement Two-dimensional parity check code to detect two or more-bit error during the message transmission from sender to receiver.

Code:

```c
#include <stdio.h>


int main() {
    printf("Enter the size of the row:");
    int ROWS, COLS;
    scanf("%d", &ROWS);
    printf("Enter the size of the columns");
    scanf("%d", &COLS);

    int data[ROWS][COLS];
    int row_parity[ROWS];
    int col_parity[COLS];
    int total_parity = 0;

    // Input the data matrix
    printf("Enter the data matrix:\n");
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            scanf("%d", &data[i][j]);
        }
    }

    // Calculate row parities
    for (int i = 0; i < ROWS; i++) {
        row_parity[i] = 0;
        for (int j = 0; j < COLS; j++) {
            row_parity[i] ^= data[i][j];
        }
        total_parity ^= row_parity[i];
    }

    // Calculate column parities
    for (int j = 0; j < COLS; j++) {
        col_parity[j] = 0;
        for (int i = 0; i < ROWS; i++) {
            col_parity[j] ^= data[i][j];
        }
        total_parity ^= col_parity[j];
    }

    // Display the data matrix
    printf("\nData Matrix:\n");
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", data[i][j]);
        }
        printf("| %d\n", row_parity[i]);
    }
    for (int j = 0; j < COLS; j++) {
        printf("--");
    }
    printf("\n");
    for (int j = 0; j < COLS; j++) {
        printf("%d ", col_parity[j]);
    }
    printf("| %d\n", total_parity);

    // Check if any error is detected
    if (total_parity == 0) {
        printf("\nNo error detected.\n");
    } else {
        printf("\nError detected.\n");
    }

    return 0;
}
```

Output:



```
"C:\Users\jerin\Documents\Computer Networks Clion\simple_parity.exe"
Enter the size of the row:4
Enter the size of the columns4
Enter the data matrix:
1
0
0
1
0
0
1
0
1
1
0
1
0
0
1
1

Data Matrix:
1 0 0 1 | 0
0 0 1 0 | 1
1 1 0 1 | 1
0 0 1 1 | 0
--------
0 1 0 1 | 0

No error detected.

Process finished with exit code 0
```

Working:

- The data is entered according to the user's choice based upon the size of each data and the number of data.
- This data is then arranged in rows and columns, in a matrix form.
- It then checks if the number of 1's in each row is odd or even. If it is even, then 0 is added in the end of row, in an extra column. Else, 1 is added in the extra column.
- Then the same is done vertically for each column. It checks if the number of 1's in each column is even or odd. If it is odd, then 1 is added in an extra row at the end, else 1 is added to the extra row.
- These new row and column will be the parity row and the parity column.
- The receiver will check if the data matches with the parity row and the parity column. That is, if the rows and columns adds up to parity row and column.

Question 3:

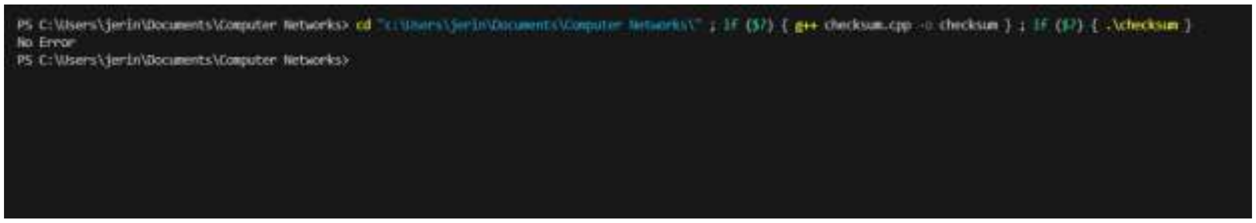
Write the C/C++ Program to implement Checksum technique to detect error during the message transmission from sender to receiver.

Code:

```cpp
// C++ implementation of the above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the One's complement
// of the given binary string
string Ones_complement(string data)
{
    for (int i = 0; i < data.length(); i++) {
        if (data[i] == '0')
            data[i] = '1';
        else
            data[i] = '0';
    }

    return data;
}

// Function to return the checksum value of
// the given string when divided in K size blocks
string checkSum(string data, int block_size)
{
    // Check data size is divisible by block_size
    // Otherwise add '0' front of the data
    int n = data.length();
    if (n % block_size != 0) {
        int pad_size = block_size - (n % block_size);
        for (int i = 0; i < pad_size; i++) {
            data = '0' + data;
        }
    }

    // Binary addition of all blocks with carry
    string result = "";

    // First block of data stored in result variable
    for (int i = 0; i < block_size; i++) {
        result += data[i];
    }

    // Loop to calculate the block
    // wise addition of data
    for (int i = block_size; i < n; i += block_size) {

        // Stores the data of the next block
        string next_block = "";

        for (int j = i; j < i + block_size; j++) {
            next_block += data[j];
        }

        // Stores the binary addition of two blocks
        string additions = "";
        int sum = 0, carry = 0;

        // Loop to calculate the binary addition of
        // the current two blocls of k size
        for (int k = block_size - 1; k >= 0; k--) {
            sum += (next_block[k] - '0')
                   + (result[k] - '0');
            carry = sum / 2;
            if (sum == 0) {
                additions = '0' + additions;
                sum = carry;
            }
            else if (sum == 1) {
                additions = '1' + additions;
                sum = carry;
            }
            else if (sum == 2) {
                additions = '0' + additions;
                sum = carry;
            }
            else {
                additions = '1' + additions;
                sum = carry;
            }
        }

        // After binary add of two blocks with carry,
        // if carry is 1 then apply binary addition
        string final = "";

        if (carry == 1) {
            for (int l = additions.length() - 1; l >= 0;
                 l--) {
                if (carry == 0) {
                    final = additions[l] + final;
                }
                else if (((additions[l] - '0') + carry) % 2
                         == 0) {
                    final = "0" + final;
                    carry = 1;
                }
                else {
                    final = "1" + final;
                    carry = 0;
                }
            }

            result = final;
        }
        else {
            result = additions;
        }
    }
```

```cpp
    // Return One's complements of result value
    // which represents the required checksum value
    return Ones_complement(result);
}

// Function to check if the received message
// is same as the senders message
bool checker(string sent_message,
             string rec_message,
             int block_size)
{

    // Checksum Value of the senders message
    string sender_checksum
            = checkSum(sent_message, block_size);

    // Checksum value for the receivers message
    string receiver_checksum = checkSum(
            rec_message + sender_checksum, block_size);

    // If receivers checksum value is 0
    if (count(receiver_checksum.begin(),
              receiver_checksum.end(), '0')
        == block_size) {
        return true;
    }
    else {
        return false;
    }
}

// Driver Code
int main()
{
    string sent_message
            = "10000101011000111001010011101101";
    string recv_message
            = "10000101011000111001010011101101";
    int block_size = 8;

    if (checker(sent_message,
                recv_message,
                block_size)) {
        cout << "No Error";
    }
    else {
        cout << "Error";
    }

    return 0;
}
```

Output:



```
PS C:\Users\jerin\Documents\Computer Networks> cd "c:\Users\jerin\Documents\Computer Networks\" ; if ($?) { g++ checksum.cpp -o checksum } ; if ($?) { .\checksum }
No Error
PS C:\Users\jerin\Documents\Computer Networks>
```

Working:

- The Checksum function first divides the whole string into the block size determined by the user. And then the sum of all the numbers are checked using binary addition.
- The One’s Compliment function finds the 1’s compliment of the binary string.
- The Main function has two strings, one from the sender and the other from the receiver
- The checker function receives the sender’s string, receiver’s string and the checksum number and check if the data is corrupted or not, by checking if the compliment of the sender’s checksum and the receiver’s compliment is 0. If it is 0, the data is not corrupted, else it is corrupted