# Experiment 1: Error Detection using Parity

**Aim:** To apply Parity check rules for error detection

**Objective:** After carrying out this experiment, students will be able to:

* Apply 1D and 2D parity rules for error detection
* Analyze the difference between 1D and 2D parity and their limitations

**Problem statement:** You are required to write separate programs to demonstrate the use of 1D and 2D parity. Take the input bit streams from the user. Your programs should calculate the parity and display the input and output bit streams.

**Analysis:** While analyzing your program, you are required to address the following points:

* Why can this method not be used to correct errors?
* How are 1D and 2D parity different?
* What are the limitations of this method of error detection?

**MARKS DISTRIBUTION**

|  |  |  |
| --- | --- | --- |
| **Component** | **Maximum Marks** | **Marks Obtained** |
| Preparation of Document | 8 |  |
| Results | 8 |  |
| Viva | 4 |  |
| **Total** | **20** |  |

Submitted by: Subhendu Maji

Register No: 18ETCS002121

1. Algorithm/Flowchart

1D Parity

STEP 1: Start

STEP 2: input number of data bits - n

STEP 3: input data bits – input[]

STEP 4: choose 1 for Odd Parity, 2 for even parity.

STEP 5: Count total number of 1’s in the data bits.

STEP 6: For Odd parity

6.1 if count == even, then parityBit = 1

6.2 if count == odd, then parityBit = 0

STEP 7: For Even parity

7.1 if count == even, then parityBit = 0

7.2 if count == odd, then parityBit = 1

STEP 8: input[n] = parityBit

STEP 9: Output input[0]...input[n] for the data to be sent.

STEP 10: Stop.

2D Parity

STEP 1: Start

STEP 2: input size of data - n

STEP 3: input data bits – a[][]

STEP 4: Count total number of 1’s in each row and store in countR[]

STEP 5: Count total number of 1’s in each column and store in countC[]

STEP 6: function oddParity (count)

6.1 if count == even, then return 1

6.2 if count == odd, then return 0

STEP 7: function evenParity (count)

6.1 if count == even, then return 0

6.2 if count == odd, then return 1

STEP 8: choose 1 for Odd Parity, 2 for even parity.

STEP 9: For Odd Parity

9.1 for i = 0 to n:

9.2 input[i][n] = oddParity(countR[i])

9.3 input[n][i] = oddParity(countC[i])

STEP 10: For Even Parity

10.1 for i = 0 to n:

10.2 input[j][n] = evenParity(countR[j])

10.3 input[n][j] = evenParity(countC[j])

STEP 11: for i = 0 to n+1:

11.1 for j = 0 to n+1:

11.2 output input[i][j]

STEP 12: Stop.

1. Program

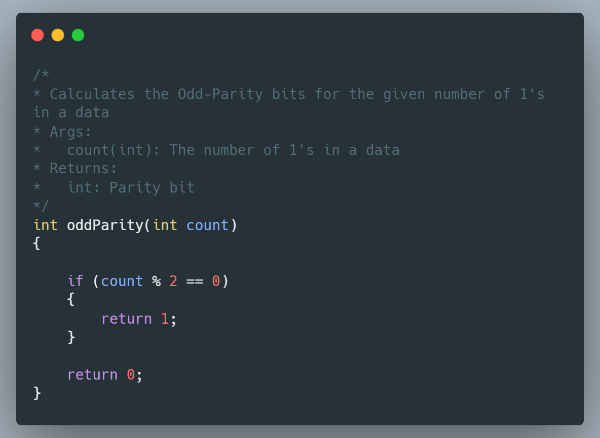


Figure 1 Function oddParity

This function takes count of 1’s as arguments and returns 1 if the count is even else returns 0.

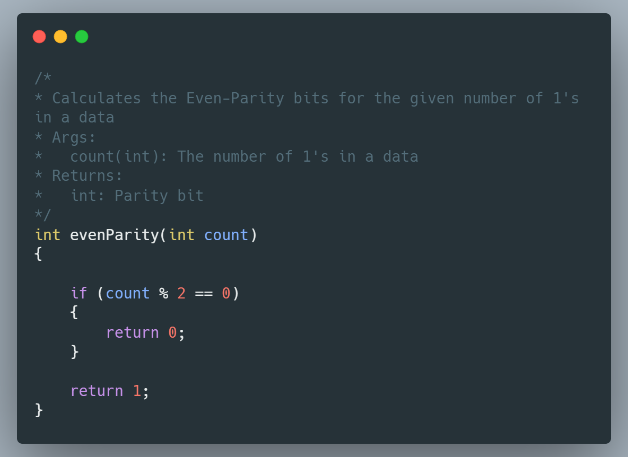


Figure 2 Function evenParity

This function takes count of 1’s as arguments and returns 0 if the count is even else returns 1.

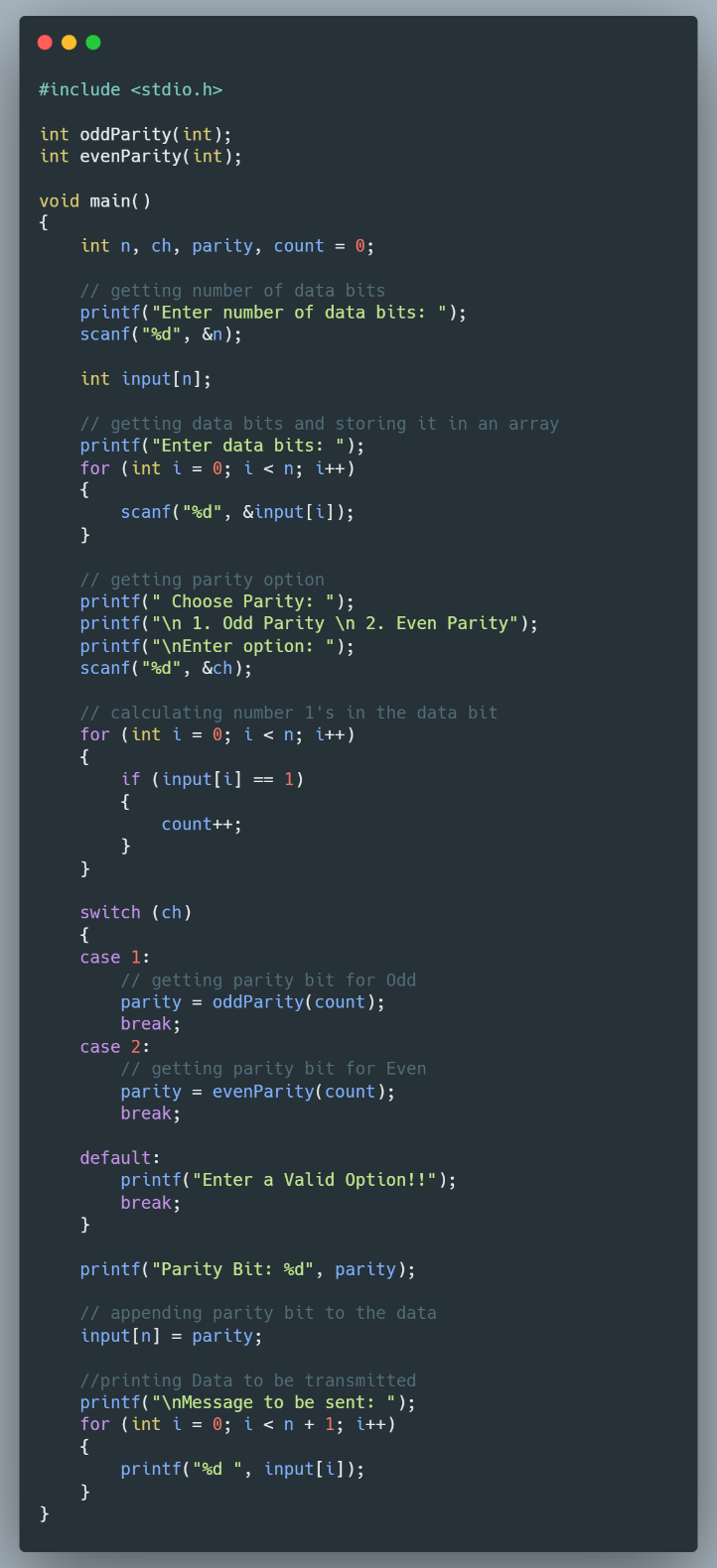


Figure 3 Source code 1D parity

The above snippet is the driver program of 1D parity check. The program first takes size of data bits and data bits as input. Then it calculates number of 1’s present in the data stream. As parity check can be odd or even, user has been given a menu option i.e. 1 for Odd parity and 2 for Even Parity. Depending on thee case odd parity or even parity switch case runs the program. For odd parity, number of 1’s is passed as argument in oddParity function (Fig. 1), and it return the parity bit which later is appended with the data stream. Same is done for even parity only evenParity function (Fig. 2) is called.

Finally, the data stream is printed with the parity bit included at the end which is our final result and should be transmitted.

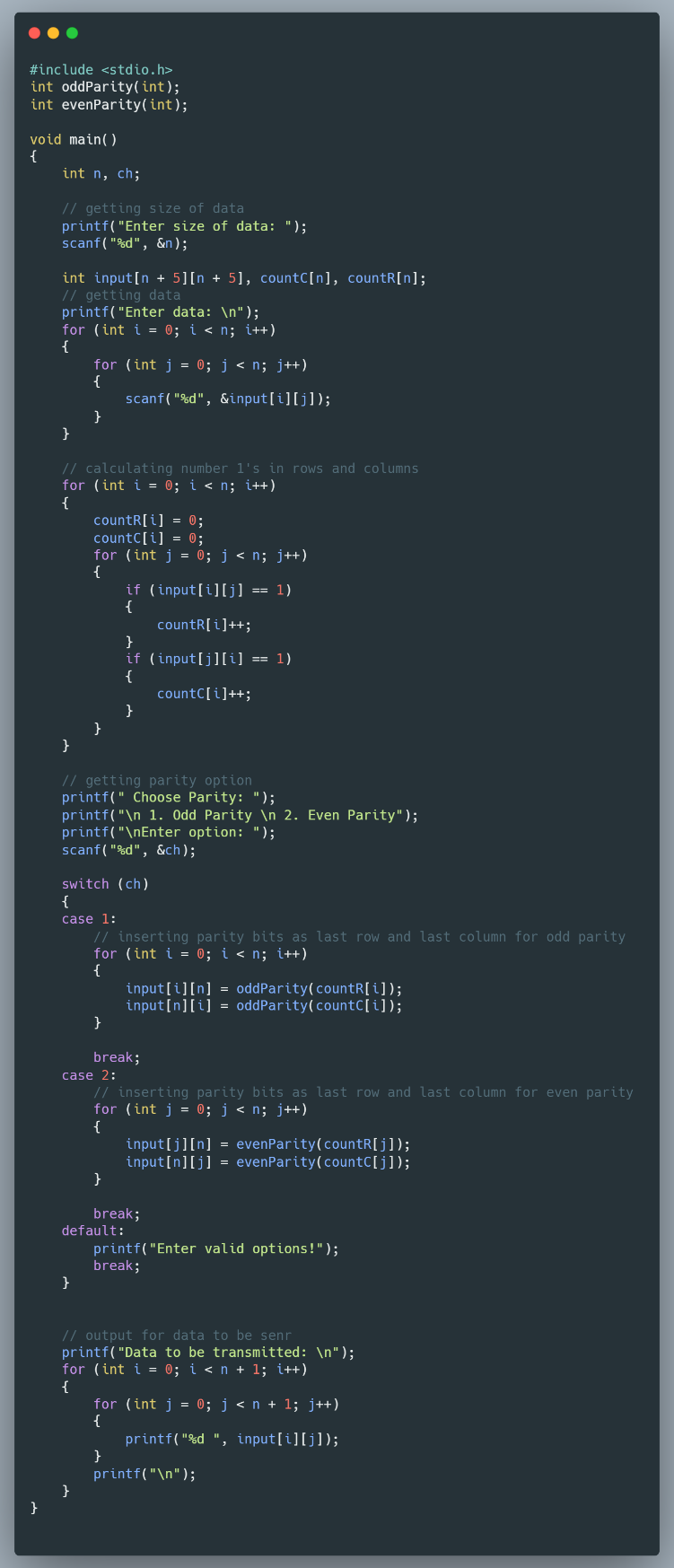


Figure 4 source code of 2D parity

The above snippet is the driver program of 2D parity check. The program first takes size of data bits and data bits as input. Then it calculates number of 1’s present in each row and column. As parity check can be odd or even, user has been given a menu option i.e. 1 for Odd parity and 2 for Even Parity. Depending on this case odd parity or even parity switch case runs the program. For odd parity, number of 1’s is passed as argument in oddParity function (Fig. 1), and it returns the parity bit which is appended at the end of the rows and columns. Same is done for even parity only evenParity function (Fig. 2) is called.

Finally, the data matrix is printed with the parity bit included at the end of each rows and column which is our final result and should be transmitted.

1. Results

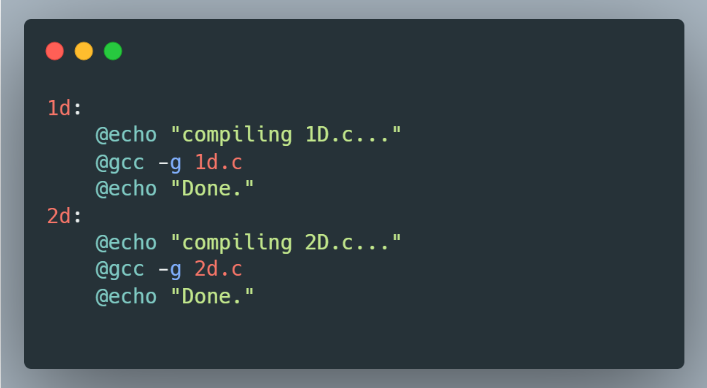


Figure 5 makefile

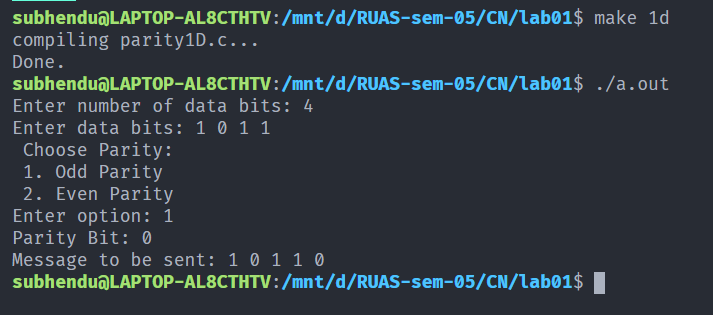


Figure 6 Execution of 1d parity

In fig. 5, the size of the data bit is 4 and data entered is 1011. The number of 1’s in the entered data is 3, which is odd. As odd parity is selected the parity bit is 0 and the data to be sent becomes 10110.

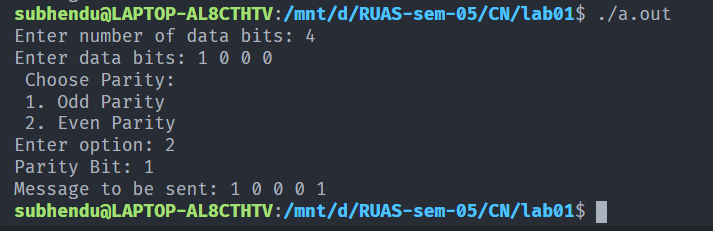


Figure 7 Execution of 1d parity

In Fig. 6, the size of the data bit is 4 and data entered is 1000. The number of 1’s in the entered data is 1, which is odd. As even parity is selected the parity bit is 1 and the data to be sent becomes 10001.

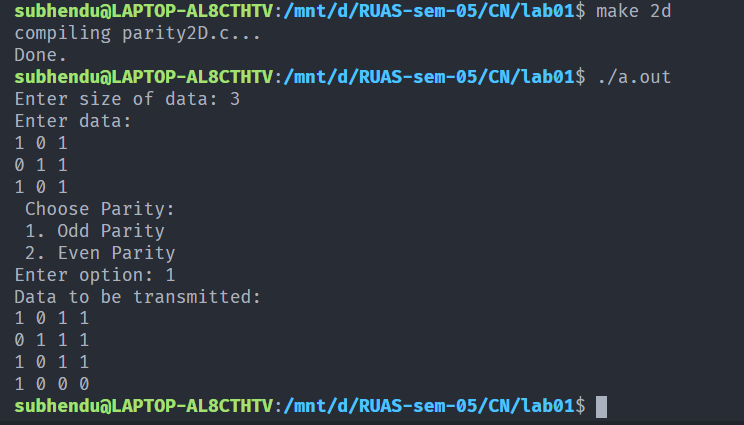


Figure 8 Execution of 2D parity

In Fig. 7, the size of the data is 3X3 matrix and the entered data is 101-011-101. As odd parity is selected, the column parity bits are 1-1-1 and row parity bits are 1-0-0.

The data to be transmitted becomes 1011-0111-1011-1000.

1. Analysis and Discussions

Errors occur when the data is being transmitted through a medium, primary culprit

being noise. To detect errors in the received data, techniques such as 1D parity and 2D

parity is used.

1D parity involves taking a packet, counting the number of 1’s in it, and appending a 0 if

there are odd number of 1s, or a 1 if there are even number of 1s, in case of odd parity,

and vice versa in case of even parity. After appending the parity bit, the packet is

transmitted through the channel and to the receiver. The receiver will recalculate the

parity bit using the received data, and if the parity bit is the same, the data is accepted,

else, it is not accepted.

2D parity is an extension of 1D parity, where the data to be sent is reshaped into a 2D matrix, and the 1D parity for each row and each column is calculated. The

row parities are interleaved with the original data, and the column parities are

appended at the end. The receiver deconstructs the received data and recalculates the

parities, accepting the data if the parities are unchanged.

1. Conclusions

The programs to calculate 1D parity and 2D parity and verify the same was successfully

implemented and executed. The advantages and limitations of the parity techniques was

learned.

1. Comments
   1. Limitations of the experiment

* Parity cannot be used to correct errors since it cannot find which bit was corrupted.
* 1D parity cannot detect burst errors
* In 1D parity, if two bits change in such a way that the parity bit remains unchanged, the data is classified as valid despite being erroneous.
  1. Limitations of the results obtained
* 2D parity involves appending a large number of redundant bits when compared to 1D parity, thus increasing
  1. Learning
     + The 1D and 2D parity techniques and the implementation of these techniques was learned.