Parity Check

A parity check is a simple error detection mechanism used in digital communication and data storage to detect errors in transmitted or stored data. It ensures that the number of bits with a value of 1 in a binary sequence is either even or odd, depending on the type of parity used.

There are two types of parity:

1. **Even Parity**: Ensures that the total number of 1 bits in the data (including the parity bit) is even.
2. **Odd Parity**: Ensures that the total number of 1 bits in the data (including the parity bit) is odd.

How Parity Check Works

- A **parity bit** is added to the original binary data to enforce the selected parity (even or odd).
- After data is transmitted or stored, a parity check is performed by counting the 1 bits in the received data (including the parity bit).
    - For **even parity**, the number of 1s should be even.
    - For **odd parity**, the number of 1s should be odd.
- If the parity condition is violated, an error is detected, indicating that the data may have been corrupted during transmission or storage.

Example

Let’s assume we are transmitting the following 7-bit binary data: **1011001**.

1. **Count the number of 1 bits**: There are four **1s** in the data.
2. **Apply even parity**: Since there are an even number of **1s**, the parity bit is set to **0** to maintain even parity.
3. Data to transmit: **10110010** (the parity bit 0 is added at the end).

When the data is received, the system checks the number of 1s:

- If the number of **1s** is even, the data is considered correct.
- If the number of **1s** is odd, an error is detected.

Use of Parity Check

Parity checks are commonly used in:

- **Data transmission protocols**: To ensure data integrity when sending bits over a communication channel.
- **Memory systems**: Error-detecting memory like Parity RAM uses parity bits to detect errors in stored data.
- **Magnetic storage and hard drives**: To ensure that stored data hasn't been corrupted.

Disadvantages:

- **Limited error detection**: Parity checks can only detect single-bit errors (if one bit flips). It cannot detect multi-bit errors, where two or more bits are changed.
- **No error correction**: Parity checks only detect errors but cannot correct them. For correction, more advanced techniques like Hamming codes.

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

int check_parity(uint8_t num) {
    int count = 0;

    // Count the number of 1 bits
    while (num) {
        count += num & 1;
        num >>= 1;
    }

    // If the count of 1 bits is even, the parity is correct
    return (count % 2 == 0);
}

int main() {
    char binary[9]; // 8 bits + 1 space for the null terminator
    uint8_t num = 0;

    printf("Enter an 8-bit binary number: ");
    scanf("%8s", binary);

    // Validate that the input is exactly 8 characters long and contains
only '0' or '1'
    if (strlen(binary) != 8 || strspn(binary, "01") != 8) {
        printf("Invalid input! Please enter an 8-bit binary number (only 0
and 1 characters).\n");
        return 1;
    }

    // Convert the binary string to an 8-bit unsigned integer
    for (int i = 0; i < 8; i++) {
        num <<= 1;                // Shift bits to the left by one position
        if (binary[i] == '1') {
            num |= 1;             // Set the lowest bit if the current
character is '1'
        }
    }

    if (check_parity(num)) {
```

```
        printf("Parity is correct (even number of 1 bits).\n");
    } else {
        printf("Parity is incorrect (odd number of 1 bits).\n");
    }

    return 0;
}
```