

Data Communication & Networking Lab File

(Sem V)  
  
Experiment- 3

Write a program for CRC and Hamming code.

|  |  |
| --- | --- |
| Name | Aaradhya Bhatiya |
| SAP ID | 500108359 |
| Roll no. | R2142221035 |
| Course | BTech CSE G&G (2022-26) |

Submitted to: Prof. Alok Jhaldiyal

**Experiment 3**

**Write a program for CRC and Hamming code.**

**Aim:** Study & implementation of the concept of CRC and Hamming code.

**Apparatus (Software):** Any programming language.

**Theory:** A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data.

**Hamming code** is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver.

**Procedure:** Understand the concept of CRC and Hamming code and write the program.

**Conclusion:** Gain the knowledge about the CRC and Hamming code.

**Algorithm**

**Cyclic Redundancy Check (CRC):** **Input**:

* Data: Binary data to be transmitted.
* Divisor: Binary representation of the divisor (polynomial).

 **Append Zeros**:

* Append n zeros to the data, where n is the length of the divisor minus one.

 **Division (Binary XOR)**:

* Perform binary division (XOR operation) between the data (with appended zeros) and the divisor.
* Continue performing XOR until the entire data is processed.
* The remainder after the division is called the **CRC remainder**.

 **Generate Codeword**:

* The codeword is the original data appended with the CRC remainder.

 **Transmission**:

* Send the codeword (data + CRC remainder) to the receiver.

 **Error Detection at Receiver**:

* Perform the same division (binary XOR) on the received codeword using the divisor.
* If the remainder is zero, the data is correct (no errors).
* If the remainder is non-zero, an error occurred during transmission.

**Hamming code:** **Input**:

* Data Word: The binary data to be transmitted.

 **Determine Number of Parity Bits**:

* Calculate the number of parity bits p required using the formula: 2^p ≥ n + p + 1
  + where n is the length of the data word.

 **Position the Parity Bits**:

* Insert parity bits at positions that are powers of 2 (1st, 2nd, 4th, 8th positions, etc.) in the code word.
* Fill other positions with the data bits.

 **Calculate Parity Bits**:

* For each parity bit, check its corresponding positions (as determined by binary representation of positions).
* Calculate the parity (even parity or odd parity) based on the values of the bits at those positions.
* Set the parity bits accordingly.

 **Generate Codeword**:

* The codeword is the data bits combined with the parity bits.

 **Transmission**:

* Send the codeword to the receiver.

 **Error Detection at Receiver**:

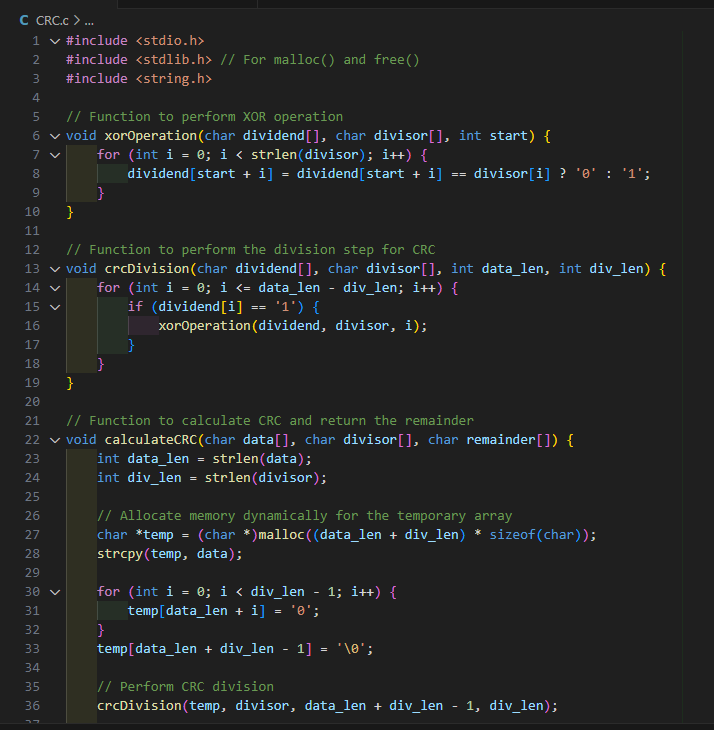
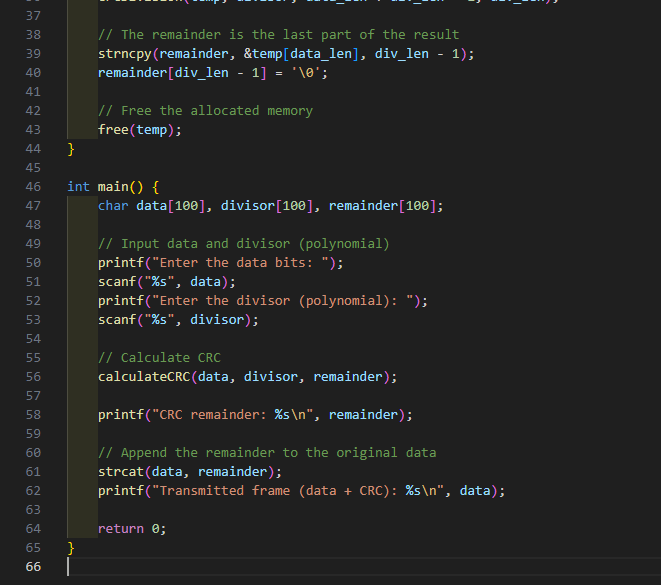
* At the receiver’s end, re-calculate the parity bits.
* If all recalculated parity bits are zero, the data is correct.
* If some recalculated parity bits are non-zero, use the binary positions of those errors to detect and correct the bit error in the codeword.

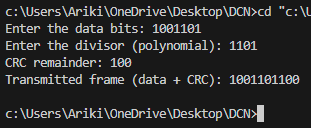
 **Error Correction**:

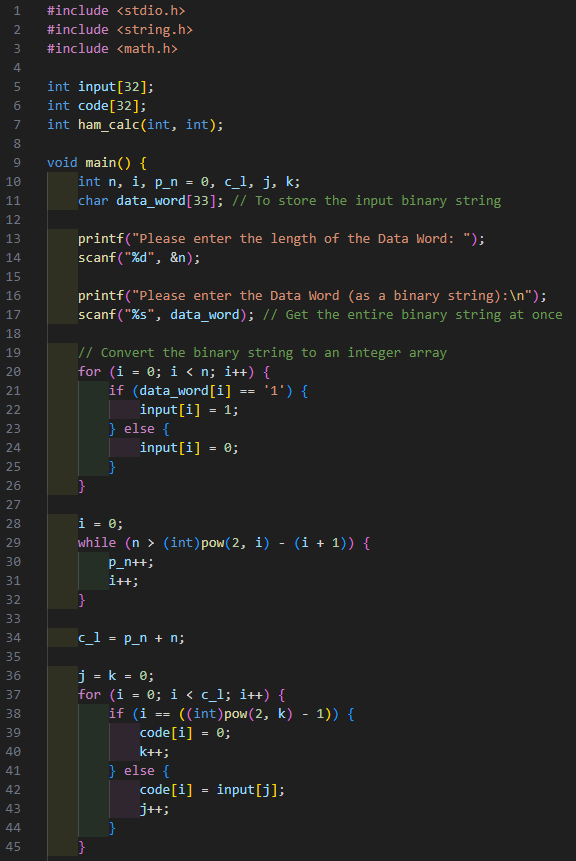
* Identify the bit position where the error occurred using the positions of the erroneous parity bits.
* Flip the bit at that position to correct the error.

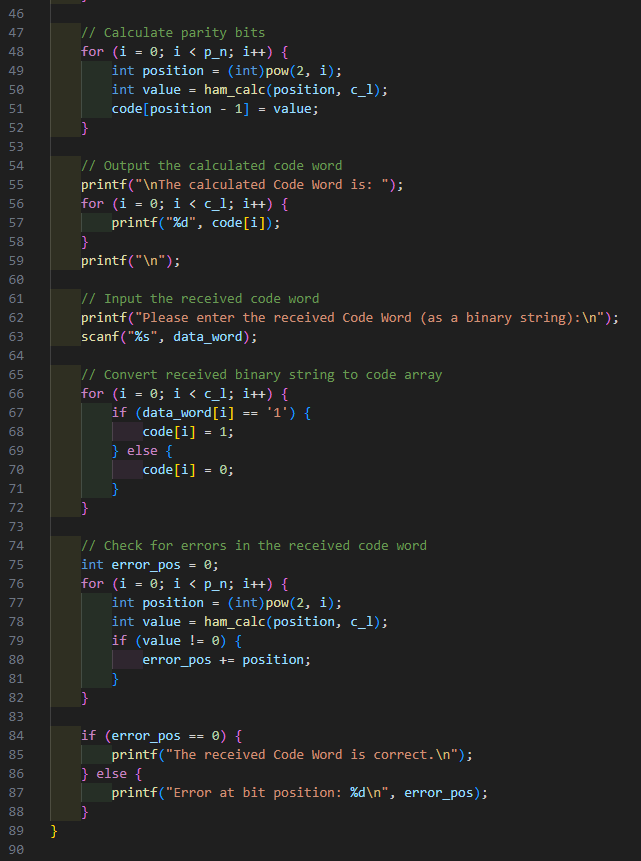
**Code:**

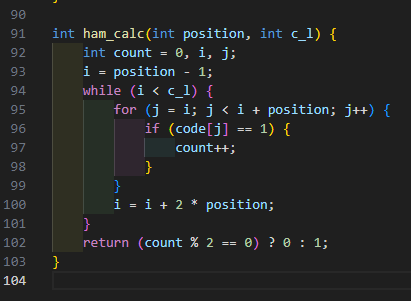
CRC:

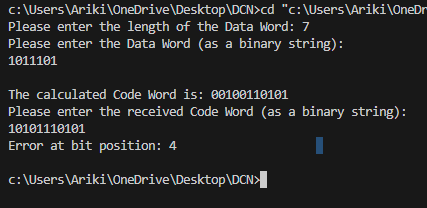
Output:  


Hamming code: ****





Output:

****