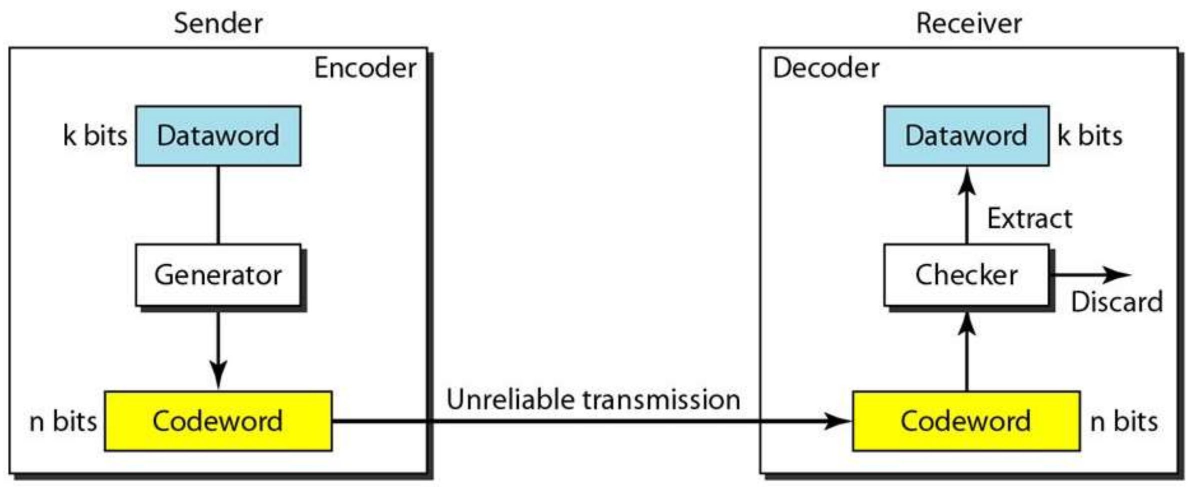
Data Communication Assignment 1
Instructor: Boonsit Yimwadsana

Submission deadline: April 26, 2019
This assignment can be done in a group of one or two students.

In data communication, error can happen during the transmission of data from sender to receiver over unreliable medium.



In order to help the receiver check for error, redundancy bits are added to the dataword to help the receiver verifies the validity of the received data (codeword).

Use the knowledge you learnt in class, implement the following programs or functions and write a demo code showing how all these functions can be used for error control.

1. **Unreliable transmission**: out_frame = *unreliable_transmission*(in_frame)
   Implement unreliable transmission function which simulates an unreliable transmission link between a sender and the receiver. This function is described by its prototype as follows:

   Input:
   1. bit string of any size up to k (input frame).
   2. Probability of having a bit in error (p).

   Output:
   1. bit string of size k defined by the input bit string (output frame).

   Behavior:

The function randomly generates errors on the bits of the input frame. A bit can be in error with probability p. The function outputs output frame which could be a corrupted or uncorrupted frame.

2. **Parity bit**:

Generator: codeword = *parity_gen*(dataword, parity_type, size)

Implement a function that generates one or two-dimensional even parity codeword to an input dataword.

Input:
1. A bit string of any size up to k (input frame) where k ≥ 9 (input frame)
2. Type of parity (even, odd, two-dimensional-even, two-dimensional-odd)
3. Size of two-dimensional block for dataword (if two-dimensional parity bit is used).

Output:
1. A code word based on the type of parity

Behavior

This function calculates the redundancy bit(s) for the dataword and output the codeword. If two-dimensional parity check is used, the size of the two-dimensional block for dataword must be specify. For example, 3x3, 4x4.

Checker: validity = *parity_check*(codeword, parity_type, size)
Input:
1. A bit string of any size up to k (codeword)
2. Type of parity (even, odd, two-dimensional-even, two-dimensional-odd)
3. Size of two-dimensional block for dataword (if two-dimensional parity bit is used).

Output:
1. Validity of codeword

Behavior

This function verifies the codeword.

3. **CRC**

Generator: codeword = *CRC_gen*(dataword, divisor)

Implement a function that generates CRC codeword for an input dataword.

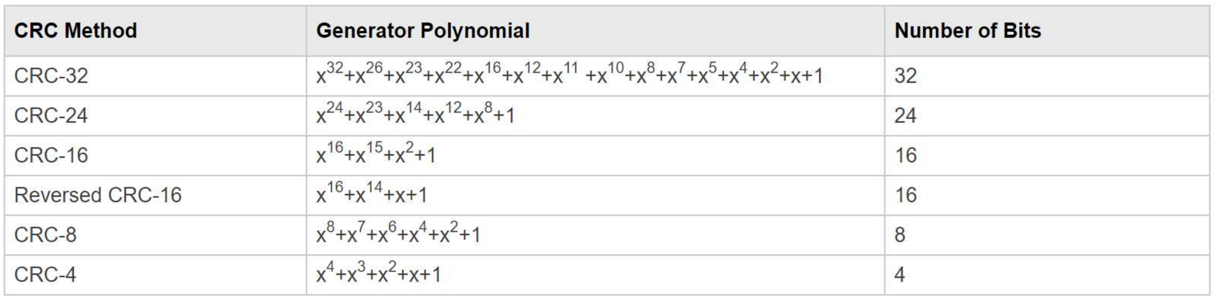
Input:

1. Dataword of size k
2. Divisor

Output:
1. A codeword based on CRC

Behavior

This function calculates the redundancy bit(s) for the dataword and output the codeword for CRC. Hint: you have to create a modulo-2 division function. For example, use CRC divisor as follows:



CRC Method	Generator Polynomial	Number of Bits
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$	32
CRC-24	$x^{24}+x^{23}+x^{14}+x^{12}+x^{8}+1$	24
CRC-16	$x^{16}+x^{15}+x^{2}+1$	16
Reversed CRC-16	$x^{16}+x^{14}+x+1$	16
CRC-8	$x^{8}+x^{7}+x^{6}+x^{4}+x^{2}+1$	8
CRC-4	$x^{4}+x^{3}+x^{2}+x+1$	4

Checker: validity = *CRC_check*(codeword, divisor)

Input:
1. codeword size k
2. divisor

Output:
1. Validity of codeword

Behavior

This function verifies the CRC codeword.

4. Checksum

Generator: codeword = *Checksum_gen*(datawords, word_size, num_blocks)

Implement a function that generates a checksum codeword for a set of input datawords.

Input:
1. A set of datawords
2. Size of each dataword (word_size)
3. Number of datawords used (num_blocks)

Output:
1. A codeword based on Checksum

Behavior

This function calculates the redundancy bit(s) for the dataword and output the codeword for Checksum.

Checker: validity = *Checksum_check*(codeword, word_size, num_blocks)
Input:
1. codeword
2. Size of each dataword (word_size)
3. Number of datawords used (num_blocks)

Output:
1. Validity of codeword

Behavior

This function verifies the CRC codeword.

5. Hamming code

Generator: codeword = *Hamming_gen*(dataword)

Implement a function that generates a codeword for an input dataword.

Input:
1. A dataword

Output:
1. A codeword based on Hamming code

Behavior

This function calculates the redundancy bit(s) for the dataword and output the codeword for Hamming code.

Checker: error_pos = *Hamming_check*(codeword)
Input:
1. codeword

Output:
1. Position of error (in case of a single bit error)

Behavior

This function verifies the Hamming codeword and report the location of error for the case of single bit error.