CRC IMPLEMENTATION

Key points of the program

The task implements the Cyclic Redundancy Check (CRC), implemented in C++.

The data to be transmitted consists of a fixed number of bits.

The user is asked to provide the calculation polynomial of the Frame Check Sequence (FCS), variable length.

A number of Frames are created to be transmitted, with random data. For each frame, its FCS is calculated.

Each frame is then transmitted to the recipient. During transmission, any corruption in the data due to noise is simulated. In particular, each bit has a certain probability of being tampered with during transmission. This Bit Error Rate (BER) is specific and affects each bit equally. This procedure is repeated for the transmission of FCS, in the same way. The number of packets transmitted by error is recorded, either in the frame itself or in the FCS.

The CRC then simulates the recipient's control of the data received, performing the same process as the sender.

The number of packets in which the CRC detected the error is recorded, as well as those in which the corrupted packets failed to detect.

Finally, the aggregate results are printed.

Brief explanation of code functions

int main()

The use in binary form gives the polynomial for the CRC in which a value check is performed.

The polynomial is converted to decimal form.

Frames are created to transmit *k* bits. *k* can be changed, so transmission and variable length control of data are supported.

Each frame is randomly generated with an equal probability that each bit is 0 or 1.

The FCS of the frame is calculated.

Each bit of the frame, based on the BER probability, can be changed from 0 to 1 or from 1 to 0 (XOR 1 bit).

The same is repeated for FCS broadcasting.

If even 1 bit of the data was corrupted, it is counted as a packet received by error.

The CRC is running, and if the check was not successful, it is counted as an error detection. In case the check was successful even though the packet was corrupted, it is counted as an error that was not detected.

The aggregate data and the resulting percentages are printed.

unsigned int fcs(long long n, unsigned int k, unsigned long long p, unsigned int CRC)

The function calculates the FCS of *the data n*, based on the polynomial *p*, and returns the FCS which consists of *CRC* bits.

It uses the following variables:

bitstream A copy of the data to be transmitted *n* in an unmarked integer so that it is considered a bitstream and slips with non-unexpected results.

Note: Although all slips are left-handed, so they are not affected by any sign, it is nevertheless used for the correctness of logic – although not necessary.

mask Pointer to the data to be transmitted *n*, which constantly slides to the right, pointing to the bit to be appended to the *op* operator. It acts as a mask for bitwise operations.

msb Pointer on the MSB of *the op*. It acts as a mask to check if it is 1, so the XOR can be executed.

It creates the 1st operator of the XOR operation from *n*, using the *mask* mask, in order to get the *CRC+1* bits required one by one.

Performs sequential XOR operations until all bits of n are used.

Then as many 0s as the *CRC are added to the operator*. This addition does not take place at the beginning, so that any digits of n are not lost during the slide.

Sequential XOR operations continue until there are no more bits available and no more XOR can be executed as the op's MSB is not 1.

FCS is returned.

Special mention was given that the operations in the function should be bitwise (slips and logical operations) and not numerical, in order to achieve speed in control and reduce the required computing power.

bool crc(long long n, unsigned int k, unsigned long long p, unsigned int CRC,

unsigned int FCS)

The function checks whether the data was transmitted unchanged or there were errors during transmission.

Its operation is identical to that of *fcs*.

They differ in the fact that in *the crc* they are not added at the end of *n* zeros, but the FCS that has been transmitted.

Returns *true* if the check was successful, *false*, otherwise.

The operations are also bitwise.

void perform_xor(unsigned long long n, unsigned long long p, unsigned int& op, unsigned long long& mask, unsigned long long msb)

The function performs the XOR operations. It was implemented to reuse code.

It works as follows:

If there is a bit available in the *data to be* transmitted n and the MSB of the *op* operator is not 1, append it to the end of the *op operator*.

If MSB is 1, it executes the XOR between *op* and polynomial *p*.

Repeats the process as long as there are available bits in *n*.

Examples of operation

For *k=32, BER=1/*1,000, *p=110101*, 1,000,000 frames

Errors in transmission:	36770	3.677% of frames
CRC detected errors:	36739	3.674% of frames, 99.916% of errors
CRC undetected errors:	31	0.084% of errors

For *k=32, BER=1/10,000, p=110101*, 1,000,000 frames

Errors in transmission:	4489	0.449% of frames
CRC detected errors:	4489	0.449% of frames, 100.000% of errors
CRC undetected errors:	0	0.000% of errors

For *k=32*, *BER=1/1,000, p=110101101*, 1,000,000 frames

Errors in transmission:	39452	3.945% of frames
CRC detected errors:	39447	3.945% of frames, 99.987% of errors
CRC undetected errors:	5	0.013% of errors

For *k=64, BER=1/1,000, p=110101*, 1,000,000 frames

Errors in transmission:	67179	6.718% of frames
CRC detected errors:	67053	6.705% of frames, 99.812% of errors
CRC undetected errors:	126	0.188% of errors

For *k=20, BER=1/1,000, p=110101, 1,000,000 frames*

```
1000000 frames transmited
Errors in transmission: 25028   2.503% of frames
CRC detected errors:    25016   2.502% of frames, 99.952% of errors
CRC undetected errors:  12      0.048% of errors
```