Exercise 1: Binary Arithmetic Encoder/Decoder

The classes ArithEnc and ArithDec implement a binary arithmetic encoder and decoder, respectively. Make yourself familiar with the interface of these classes.

- The function `start()` starts the arithmetic encoder/decoder and has to be called at the beginning of a picture.
- The function `finish()` terminates the arithmetic codeword. It has to be called at the end of a picture.
- The functions `encodeBinEP()` and `decodeBinEP()` arithmetically encode/decode a bin (flag) with a fixed probability mass function of \((0.5, 0.5)\). This is similar to writing/reading the bin/flag directly to/from the bitstream.
- The functions `encodeBinsEP()` and `decodeBinsEP()` arithmetically encode/decode multiple bins (flags) with a fixed probability mass function of \((0.5, 0.5)\). This is similar to writing/reading the bins/flags directly to/from the bitstream.

⇒ The main functionality is provided by the functions `encodeBin()` and `decodeBin()`. These function encode/decode a bin/flag with an adaptive probability model.

- The probability model is provided by the class `ProbModel`.
- During encoding and decoding, the probability models are updated automatically.
Exercise 2: Implement Non-Adaptive Arithmetic Coding

Encode and decode all bins/bits using non-adaptive binary arithmetic coding (i.e., arithmetic coding with fixed pmfs of (0.5, 0.5)).

- Add instances of the classes ArithEnc and ArithDec to the classes EntropyEncoder and EntropyDecoder, respectively.
- Start and finish the arithmetic encoder/decoder at the start/end of the entropy encoding/decoding for a picture.
- In the entropy encoder, replace
  - all occurrences of OBitstream::addBit() with ArithEnc::encodeBinEP()
  - all occurrences of OBitstream::addBits() with ArithEnc::encodeBinsEP()
- In the entropy decoder, replace
  - all occurrences of IBitstream::getBit() with ArithDec::decodeBinEP()
  - all occurrences of IBitstream::getBits() with ArithDec::decodeBinsEP()

Test your implementation:
- Verify that encoder and decoder work correctly
- Test the coding efficiency (in relation to the previous version)
Exercise 3: Implement Adaptive Arithmetic Coding

Upgrade the arithmetic coding by using adaptive probability models

- Successively replace the non-adaptive coding (functions `encodeBinEP()` and `decodeBinEP()`) with an adaptive coding (functions `encodeBin()` and `decodeBin()`)

- Use a separate probability model for the following classes of bins:
  - The coded block flags
  - The first bin of the x/y coordinates for the last significant coefficient
  - The bins of the unary prefix part for the x/y coordinates
  - The first bin (significance bin) of the quantization indexes (levels)
  - The bins of the unary prefix part for the quantization indexes (levels)

- The suffix part of the ExpGolomb code and the sign bits should be coded with fixed non-adaptive probabilities (i.e., do not modify these parts)

Test your implementation:

- Verify that encoder and decoder work correctly
- Test the coding efficiency (in relation to the non-adaptive version)
Exercise 4: Further Improve the Arithmetic Coding

The efficiency of the arithmetic coding depends on the usage of adaptive probability models. Basically, different probability models should be used for bins that are expected to have different probabilities.

- Think about how we could further improve the coding efficiency.
- Try some ideas for improving coding efficiency.