Exercise 1: Implementation of DCT-II (General Goal)

- Implement a class Transform

```cpp
class Transform
{
public:
  Transform();
  ~Transform() {}
  
  void fwdTransform( Block& blk, int log2BlkSize ) const; // 2d forward DCT-II
  void invTransform( Block& blk, int log2BlkSize ) const; // 2d inverse DCT-II

private:
  ...
};
```

- The function `fwdTransform()` shall apply the 2d DCT-II to an input block of samples. As result, all samples should be replaced with the resulting transform coefficients.

- The function `deQuantize()` shall apply the inverse 2d DCT-II to an input block of transform coefficients. As result, the block shall contain the resulting samples.

- For simplifying the implementation, the input parameter `log2BlkSize` specifies that binary logarithm of the block width and height.

  Similar as for scalar quantization, we want to approximate all calculations using integer arithmetic (see following slides).
Exercise 1a: Integer Approximation of Transform Matrices

The basis vectors $b_k$ ($0 \leq k < N$) of the real-valued DCT-II of size $N$ are given by the vector elements (see lecture) $b_{ki}$ with $0 \leq i < N$,

$$b_{ki} = \frac{a_k}{\sqrt{N}} \cdot \cos \left( \frac{\pi}{N} \cdot k \cdot \left( i + \frac{1}{2} \right) \right)$$

$$a_k = \begin{cases} 1 & : k = 0 \\ \sqrt{2} & : k > 0 \end{cases}$$

The basis vectors $b_k$ form the rows of the forward $N \times N$ transform matrix and the rows of the inverse $N \times N$ transform matrix.

We want to use integer transform matrices $B^{(int)} = \{ b_{ki}^{(int)} \}$, for which all matrix elements are given by scaled and rounded DCT coefficients according to

$$b_{ki}^{(int)} = \text{round} \left( 2^6 \cdot b_{ik} \right)$$

→ Determine the integer transform matrices for all supported block sizes (2, 4, 8, 16, 32, 64, 128) in the constructor `Transform::Transform()`

→ Use the pre-calculated integer matrices for implementing the actual transforms

→ Note: The scaling of the transform matrices has to be compensated by an inverse scaling after the horizontal and vertical transform. The function `Block::scale( .. )` can be used for doing the required scaling.
Exercise 1b: Transform Implementation

The 2d DCT-II (and also its inverse transform) consists of a horizontal and vertical transform.

Note that the transform could be implemented by the following steps:

- Horizontal transform
- Transposition of resulting block
- Horizontal transform (same as in the first step)
- Transposition of resulting block (same as in second step)

Hence, a **clever implementation** would only require a single transform function:

```cpp
void horTrafoTranspose( const Block& in, Block& out, transform matrix ) const
{
    // apply horizontal transform to input block "in"
    // and store transposed output in "out"
}
```

in which the result of a horizontal transform is directly stored in a transposed output block.
Exercise 1c: Application and Testing of Transform Coding

- Implement transform coding of sample blocks
- Add an instance of the class Transform to the class BlockCoder
- Use the implemented transforms in the functions BlockCoder::compressBlock() and BlockCoder::reconstructBlock()

Testing

- Test the effect of the transform on coding efficiency
- Compare the coding efficiency (rate-distortion curves) for the following versions
  - our codec without a transform (previous version)
  - our codec with the DCT-II transform
    for all test images and the entire QP range (0..31).
- Does the transform improve coding efficiency?
Exercise 2: Reduce Effect of Intermediate Rounding

- In our current encoder, we apply two rounding steps
  - at the end of the 2d transform
  - in the quantization

→ Reduce the effect of intermediate rounding in the encoder by
  - removing the rounding at the end of the transform and
  - consider the required scaling in the quantization process

→ Implement a similar improvement for the reconstruction process

Testing

→ Check whether the modification improves coding efficiency
→ Judge the coding efficiency relative to JPEG
→ How could we improve our codec?