```
// Przemyslaw Sliwinski@2010
// Haar transforms with coefficients normalized in L2-norm, i.e. {h_n} = {1/sqrt(2),
 1/sqrt(2)};
template <typename T>
void L2_Haar_lifting_2D_transform(basic_matrix<T> &image, int M)
{
        // An L2 normalization constant
        T const sqrt_2 = std::sqrt(2.0);

        int image_size = image.height();
        for(int m = 0; m < M; ++m)
        {
                int const step = 1 << m;
                // row wise 1D transform
                for(int i = 0; i < image_size; ++i)
                {
                        for(int j = 0; j < image_size; j += 2)
                        {
                                T &s = image[i * step][j * step],
                                  &d = image[i * step][(j + 1) * step];

                                d -= s;                 // predicting a difference

                                d /= sqrt_2;

                                s *= sqrt_2;
                                s += d;                 // updating a mean
                        }
                }
                // column wise 1D transform
                for(int i = 0; i < image_size; ++i)
                {
                        for(int j = 0; j < image_size; j += 2)
                        {
                                T &s = image[j * step][i * step],
                                  &d = image[(j + 1) * step][i * step];

                                d -= s;                 // predicting a difference

                                d /= sqrt_2;

                                s *= sqrt_2;
                                s += d;                 // updating a mean
                        }
                }
                image_size >>= 1;
        }
}

// Haar transforms with coefficients normalized in L1-norm, i.e. {h_n} = {1/2, 1/2}
template <typename T>
void L1_Haar_lifting_2D_transform(basic_matrix<T> &image, int M)
{
        int image_size = image.height();
        // row wise 1D transform
        for(int m = 0; m < M; ++m)
        {
                int const step = 1 << m;
                for(int i = 0; i < image_size; ++i)
                {
                        for(int j = 0; j < image_size; j += 2)
                        {
                                T &s = image[i * step][j * step],
                                  &d = image[i * step][(j + 1) * step];

                                d -= s;         // predicting a difference
                                s += d/2;       // updating a mean
                        }
                }
```

```
                // column wise 1D transform
                for(int i = 0; i < image_size; ++i)
                {
                        for(int j = 0; j < image_size; j += 2)
                        {
                                T &s = image[j * step][i * step],
                                  &d = image[(j + 1) * step][i * step];

                                d -= s;         // predicting a difference
                                s += d/2;       // updating a mean
                        }
                }
                image_size >>= 1;
        }
}
```