$\boxed{22}$ Let $\mathcal{T}_h$ be a mesh of $(0,1)$ with maximal mesh size h, let V_h be the corresponding finite element space (using the Courant element), and let $I_h : H^1(0,1) \to V_h$ the corresponding interpolation operator. Show the interpolation error estimate

$$|v - I_h v|_{H^1(0,1)} \;\leq\; C_1\, h\, \|v''\|_{L^2(0,1)} \forall v \in C^2(0,1).$$

Hint: Analogous to the proof of the L^2-estimate in the lecture: split into element terms, transform to the reference element, estimate the interpolation error on the reference element, transform back.

$\boxed{23}$ Let $V_g = V_0 := \{v \in H^1(0,1) : v(0) = 0\}$ and let $a : V_0 \times V_0 \to \mathbb{R}$ and $F \in V_0^*$ fulfill the assumptions of the Lax-Milgram theorem. Let V_h be the Courant FE space as in Exercise $\boxed{22}$ and $V_{0h} = V_h \cap V_0$. Furthermore, let $u \in V_0$ be the solution of the variational formulation and $u_h \in V_{0h}$ the Galerkin-FE solution. Show that for a sequence of meshes $\{\mathcal{T}_h\}$ where $h \to 0$, we have

$$\|u - u_h\|_{H^1(0,1)} \;\to\; 0,$$

even if $u \notin H^2(0,1)$.

Hint: Use Céa's lemma and estimate $\quad (u - \tilde{u}) + (\tilde{u} - I_h\tilde{u}) \quad$ for $\tilde{u} \in H^2(0,1)$. Use the fact that H^2 is dense in H^1 and make the two terms sufficiently small.

Programming.
In Tutorial 4 we computed the element stiffness matrices and load vectors. Here we will first assemble these to form the "full" stiffness matrix and load vector for a pure Neumann problem, which are then modified to implement other kinds of boundary conditions. You can again use the vector data type provided on the homepage.

$\boxed{24}$ Write a function

```
void AssembleStiffnessMatrix (const Mesh& mesh, SMatrix& mat);
```

that assembles the $(n_h + 1) \times (n_h + 1)$ stiffness matrix `mat`$=K_h$ (see Exercise $\boxed{20}$) for a given mesh `mesh`$=\mathcal{T}_h$ of (0, 1).

Test your function by constructing the stiffness matrix for at least an equidistant mesh of 20 elements and printing the result to the screen.

Hint: Set $K_h = 0$, then loop over all elements. For each element, call `ElementStiffnessMatrix` and add the entries of $K_h^{(k)}$ at the correct positions of K_h.

$\boxed{25}$ Write a function

```
void AssembleLoadVector (RealFunction f, const Mesh& mesh,
                         Vector& vec);
```

that assembles the load vector `vec`=$\underline{f}_h$ for the given function `f`=$f \in C[0,\,1]$ and the given mesh.

Test your function by constructing the load vector for at least an equidistant mesh of 20 elements for the function $f(x) = 3x + 1$ and printing the result to the screen.

Hint: Set $\underline{f}_h = 0$, then loop over all elements. For each element, call `ElementLoadVector` and add the entries of the element load vector to the right places.

$\boxed{26}$ Write a function

```
void ImplementRobinBC (int i, double g, double alpha,
                       SMatrix& mat, Vector& vec);
```

to implement the Robin boundary condition

$$
\begin{aligned}
-u'(0) + \alpha\, u(0) &= g_1 \qquad \text{if } i = 0, \\
u'(1) + \alpha\, u(1) &= g_1 \qquad \text{if } i = n_h\,,
\end{aligned}
$$

for given values `g`=g_1, `alpha`=α at the boundary node x_i identified by the index `i`=$i \in \{0, n_h\}$. Compare with Exercise $\boxed{02}$ and $\boxed{09}$.

`ImplementRobinBC` must update (modify) the stiffness matrix `mat` and the load vector `vec` which were previously computed by `AssembleStiffnessMatrix` and `AssembleLoadVector`.

Test your function on the system constructed in exercises $\boxed{24}$ and $\boxed{25}$.

$\boxed{27}$ Write a function

```
void ImplementDirichletBC (int i, double g, SMatrix& mat,
                           Vector& vec);
```

which implements the Dirichlet boundary condition

$$u(x_i) \;=\; g_0$$

for a given value `g`=g at the boundary node x_i identified by the index `i`=i. The function `ImplementDirichletBC` must update the stiffness matrix `mat` and the load vector `vec`, after having applied `AssembleStiffnessMatrix`, `AssembleLoadVector`, and `ImplementRobinBC`.

Instead of *deleting* rows or columns from the matrix, you should stay with the $(n_h + 1) \times (n_h + 1)$ matrix using the following "decoupling" technique:
For $i = 0$, the resulting system should look like

$$
\begin{pmatrix}
K_{00} & 0 & \dots & 0 \\
0 & K_{11} & \dots & K_{1n_h} \\
\vdots & \vdots & \ddots & \vdots \\
0 & K_{n_h 1} & \dots & K_{n_h n_h}
\end{pmatrix}
\begin{pmatrix}
u_0 \\ u_1 \\ \vdots \\ u_{n_h}
\end{pmatrix}
=
\begin{pmatrix}
K_{00}\, g_0 \\ f_1 - K_{10}\, g_0 \\ \vdots \\ f_{n_h} - K_{n_h 0}\, g_0
\end{pmatrix},
$$

where $[K_{ij}]_{i,j=0}^{n_h}$ and $[f_i]_{i=0}^{n_h}$ are the "full" stiffness matrix and load vector *before* the call.

Test your function on the system constructed in exercises $\boxed{24}$ and $\boxed{25}$.