Step-3 introduction report

Firstly the member variables: These follow the building blocks we have outlined above in the bullet points, namely we need a Triangulation and a DoFHandler object, and a finite element object that describes the kinds of shape functions we want to use.

The second group of objects relate to the linear algebra: the system matrix and right hand side as well as the solution vector, and an object that describes the sparsity pattern of the matrix. This is all this class needs (and the essentials that any solver for a stationary PDE requires) and that needs to survive throughout the entire program.

In contrast to this, the FEValues object we need for assembly is only required throughout assembly, and so we create it as a local object in the function that does that and destroy it again at its end.

the member functions-

make\_grid(): This is what one could call a preprocessing function. As its name suggests, it sets up the object that stores the triangulation. In later examples, it could also deal with boundary conditions, geometries, etc.

setup\_system(): This then is the function in which all the other data structures are set up that are needed to solve the problem. In particular, it will initialize the DoFHandler object and correctly size the various objects that have to do with the linear algebra. This function is often separated from the preprocessing function above because, in a time dependent program, it may be called at least every few time steps whenever the mesh is adaptively refined (something we will see how to do in step-6). On the other hand, setting up the mesh itself in the preprocessing function above is done only once at the beginning of the program and is, therefore, separated into its own function.

assemble\_system(): This, then is where the contents of the matrix and right hand side are computed, as discussed at length in the introduction above. Since doing something with this linear system is conceptually very different from computing its entries, we separate it from the following function.

solve(): This then is the function in which we compute the solution U of the linear system AU=F. In the current program, this is a simple task since the matrix is so simple, but it will become a significant part of a program's size whenever the problem is not so trivial any more (see, for example, step-20, step-22, or step-31 once you've learned a bit more about the library).

output\_results(): Finally, when you have computed a solution, you probably want to do something with it. For example, you may want to output it in a format that can be visualized, or you may want to compute quantities you are interested in: say, heat fluxes in a heat exchanger, air friction coefficients of a wing, maximum bridge loads, or simply the value of the numerical solution at a point. This function is therefore the place for postprocessing your solution.

All of this is held together by the single public function (other than the constructor), namely the run() function. It is the one that is called from the place where an object of this type is created, and it is the one that calls all the other functions in their proper order. Encapsulating this operation into the run() function, rather than calling all the other functions from main() makes sure that you can change how the separation of concerns within this class is implemented. For example, if one of the functions becomes too big, you can split it up into two, and the only places you have to be concerned about changing as a consequence are within this very same class, and not anywhere else.

this general structure — sometimes with variants in spelling of the functions' names, but in essentially this order of separation of functionality.