```
1
     // -*- C++ -*-
 2
     #ifndef ELEMENT TYPES
 3
     #define ELEMENT_TYPES
 4
 5
     #include "/src/Definitions.h"
 6
 7
     namespace ElementTypes {
 8
 9
     // simplicial element: linear triangle (if Dimension=2) or linear tetrahedron (if
     Dimension=3)
10
     template<unsigned int Dimension>
     struct Simplex {
11
12
13
       static const unsigned int
                                                            NumberOfNodes = Dimension + 1;
14
       static const unsigned int
                                                         SpatialDimension = Dimension;
15
       static const unsigned int
                                                 NumberOfNodesPerBoundary = Dimension;
16
       static const unsigned int
                                                       NumberOfBoundaries = Dimension + 1;
17
       typedef Eigen::Matrix<double, SpatialDimension, 1>
                                                                             Point;
                                                                ? VTK LINE :
18
       static const VTKCellType VtkCellType = (Dimension==1)
19
                                               (Dimension == 2) ? VTK_TRIANGLE : VTK_TETRA;
2.0
2.1
       //TODO: compute shape functions in the reference configuration of the element
       (using reference coordinates)
22
       array<double, NumberOfNodes>
23
       computeShapeFunctions(const Point &point) const {
2.4
25
         ignoreUnusedVariable<Point>(point);
26
27
         // TODO: Evaluate all entries of shapeFunctions, initialized just below
28
         array<double, NumberOfNodes> shapeFunctions;
29
         if (Dimension == 1){
30
           shapeFunctions[0] = point(0);
31
           shapeFunctions[1] = 1 - point(0);
32
         }
33
         else if (Dimension == 2){
           shapeFunctions[0] = point(0);
34
35
           shapeFunctions[1] = point(1);
36
           shapeFunctions[2] = 1 - point(0) - point(1);
37
38
         if (Dimension == 3){
39
           shapeFunctions[0] = point(0);
40
           shapeFunctions[1] = point(1);
41
           shapeFunctions[2] = point(2);
42
           shapeFunctions[3] = 1- point(0) - point(1) - point(2);
43
         // Return
44
45
         return shapeFunctions;
46
47
48
       //TODO: compute shape function derivatives in the reference configuration of the
       element (using reference coordinates)
49
       array<Point, NumberOfNodes>
50
       computeShapeFunctionDerivatives(const Point &point) const {
51
52
         ignoreUnusedVariable<Point>(point);
53
54
         // TODO: Evaluate all entries of shapeFunctionsDerivatives, initialized just below
55
         // NOTE: Of course, since we're looking at the derivatives of shape functions now,
56
         //
                  we're not handling array<double, NumberOfNodes> but instead
57
         //
                  array<double, NumberOfNodes> as of now - just wanted to stress this ;-)
58
         array<Point, NumberOfNodes> shapeFunctionDerivatives;
59
         if (Dimension == 1){
60
           shapeFunctionDerivatives[0](0) = 1;
61
           shapeFunctionDerivatives[1](0) = 0;
62
63
         if (Dimension ==2){
64
           shapeFunctionDerivatives[0](0) = 1;
65
           shapeFunctionDerivatives[1](0) = 0;
66
           shapeFunctionDerivatives[2](0) = -1;
67
68
           shapeFunctionDerivatives[0](1) = 0;
69
           shapeFunctionDerivatives[1](1) = 1;
70
           shapeFunctionDerivatives[2](1) = -1;
```

```
71
72
         if (Dimension == 3){
73
           shapeFunctionDerivatives[0](0) = 1;
74
           shapeFunctionDerivatives[1](0) = 0;
75
           shapeFunctionDerivatives[2](0) = 0;
76
           shapeFunctionDerivatives[3](0) = -1;
77
78
           shapeFunctionDerivatives[0](1) = 0;
79
           shapeFunctionDerivatives[1](1) = 1;
80
           shapeFunctionDerivatives[2](1) = 0;
81
           shapeFunctionDerivatives[3](1) = -1;
82
83
           shapeFunctionDerivatives[0](2) = 0;
84
           shapeFunctionDerivatives[1](2) = 0;
85
           shapeFunctionDerivatives[2](2) = 1;
86
           shapeFunctionDerivatives[3](2) = -1;
87
88
89
         // Return
90
         return shapeFunctionDerivatives;
91
92
     };
93
94
95
     #endif //ELEMENT_TYPES
96
```