```
// -*- C++ -*-
#ifndef ELEMENT_TYPES
#define ELEMENT_TYPES

#include "/src/Definitions.h"

namespace ElementTypes {

// simplicial element: linear triangle (if Dimension=2) or linear tetrahedron (if Dimension=3)
template<unsigned int Dimension>
struct Simplex {

  static const unsigned int                           NumberOfNodes = Dimension + 1;
  static const unsigned int                        SpatialDimension = Dimension;
  static const unsigned int                NumberOfNodesPerBoundary = Dimension;
  static const unsigned int                      NumberOfBoundaries = Dimension + 1;
  typedef Eigen::Matrix<double, SpatialDimension, 1>                  Point;
  static const VTKCellType VtkCellType = (Dimension==1)   ? VTK_LINE :
                                         (Dimension == 2) ? VTK_TRIANGLE : VTK_TETRA;

  //TODO: compute shape functions in the reference configuration of the element (using reference coordinates)
  array<double, NumberOfNodes>
  computeShapeFunctions(const Point &point) const {

    ignoreUnusedVariable<Point>(point);

    // TODO: Evaluate all entries of shapeFunctions, initialized just below
    array<double, NumberOfNodes> shapeFunctions;
    if (Dimension == 1){
      shapeFunctions[0] =  point(0);
      shapeFunctions[1] = 1 - point(0);
    }
    else if (Dimension == 2){
      shapeFunctions[0] = point(0);
      shapeFunctions[1] = point(1);
      shapeFunctions[2] = 1 - point(0) - point(1);
    }
    if (Dimension == 3){
      shapeFunctions[0] = point(0);
      shapeFunctions[1] = point(1);
      shapeFunctions[2] = point(2);
      shapeFunctions[3] = 1- point(0) - point(1) - point(2);
    }
    // Return
    return shapeFunctions;
  }

  //TODO: compute shape function derivatives in the reference configuration of the element (using reference coordinates)
  array<Point, NumberOfNodes>
  computeShapeFunctionDerivatives(const Point &point) const {

    ignoreUnusedVariable<Point>(point);

    // TODO: Evaluate all entries of shapeFunctionsDerivatives, initialized just below
    // NOTE: Of course, since we're looking at the derivatives of shape functions now,
    //       we're not handling array<double, NumberOfNodes>  but instead
    //       array<double, NumberOfNodes> as of now - just wanted to stress this ;-)
    array<Point, NumberOfNodes> shapeFunctionDerivatives;
    if (Dimension == 1){
      shapeFunctionDerivatives[0](0) = 1;
      shapeFunctionDerivatives[1](0) = 0;
    }
    if (Dimension ==2){
      shapeFunctionDerivatives[0](0) = 1;
      shapeFunctionDerivatives[1](0) = 0;
      shapeFunctionDerivatives[2](0) = -1;

      shapeFunctionDerivatives[0](1) = 0;
      shapeFunctionDerivatives[1](1) = 1;
      shapeFunctionDerivatives[2](1) = -1;
    }
    if (Dimension == 3){
      shapeFunctionDerivatives[0](0) = 1;
      shapeFunctionDerivatives[1](0) = 0;
      shapeFunctionDerivatives[2](0) = 0;
      shapeFunctionDerivatives[3](0) = -1;

      shapeFunctionDerivatives[0](1) = 0;
      shapeFunctionDerivatives[1](1) = 1;
      shapeFunctionDerivatives[2](1) = 0;
      shapeFunctionDerivatives[3](1) = -1;

      shapeFunctionDerivatives[0](2) = 0;
      shapeFunctionDerivatives[1](2) = 0;
      shapeFunctionDerivatives[2](2) = 1;
      shapeFunctionDerivatives[3](2) = -1;
    }

    // Return
    return shapeFunctionDerivatives;
  }
};

}
#endif //ELEMENT_TYPES
```