```
#ifndef ELEMENT_TWO_NODE_BAR
#define ELEMENT_TWO_NODE_BAR

#include "/src/Definitions.h"
#include "/src/Utilities.h"

namespace Elements {

class Properties {
public:
  double _area, _density;
  Properties(const double area, const double density) :
    _area(area),
    _density(density){
  }
};

template<class MaterialModel>
class FiniteBar3D {

public:

  static const unsigned int     NumberOfNodes = 2;
  static const unsigned int  SpatialDimension = 3;
  static const unsigned int  DegreesOfFreedom = 3;
  static const VTKCellType        VtkCellType = VTK_LINE;

  // Typedef's using std and Eigen Classes
  typedef Matrix<double, SpatialDimension, 1>        Vector;
  typedef array<Vector, NumberOfNodes>               NodalDisplacements;
  typedef array<Vector, NumberOfNodes>               Forces;
  typedef Matrix<double,
                 DegreesOfFreedom,DegreesOfFreedom>  NodalStiffnessMatrix;
  typedef Matrix<double,
                 NumberOfNodes*DegreesOfFreedom,
                 NumberOfNodes*DegreesOfFreedom>     StiffnessMatrix;
  typedef Matrix<double, SpatialDimension, 1>        Point;

  // Typedef based on the standard Node "NodeWithID" defined in Definitions.h
  typedef NodeWithId<Point>                          Node;

  // Typedef's derived from the MaterialModel
  typedef typename MaterialModel::Strain             Strain;
  typedef typename MaterialModel::Stress             Stress;
  typedef typename MaterialModel::TangentMatrix      TangentMatrix;

  // Public Members
  array<size_t, NumberOfNodes> _nodeIds;
  Properties                   _properties;
  Point                        _X0,_X1;
    //array<Point, NumberOfNodes>   _nodePositions;
  double                        _undeformedBarLength;

  // TODO: in case you might want to store more public members - go for it!

  FiniteBar3D(const array<Node, NumberOfNodes> &  nodes         ,
              const Properties &                  properties    ,
              const MaterialModel *               materialModel ) :
      _properties   (properties)   ,
      _materialModel(materialModel){

    //ignoreUnusedVariables(nodes);

    // TODO: Define the public member _nodeIds based on information;
    //       from nodes
      _nodeIds[0]       = nodes[0]._id;
      _nodeIds[1]       = nodes[1]._id;
      //_nodePositions[nodeIndex] =nodes[nodeIndex]._position;


    // TODO: Define the public member _X0, _X1 based on information
    //       from nodes

```

```
      _X0 = nodes[0]._position;
      _X1 = nodes[1]._position;
      // TODO: Define any private members you added by yourself

      _undeformedBarLength = (_X1-_X0).norm();

    }



    // Takes displacement and returns strain
    Strain
    computeBarStrain(const NodalDisplacements & displacements) const {

      Strain strain = Strain::Zero();

      double deformedlength = ( _X1+displacements[1] - (_X0+displacements[0]) ).norm();
      strain(0)             = (deformedlength-(_X1-_X0).norm())/((_X1-_X0).norm());

      return strain;
    }



    // TODO: Complete the function computeEnergy to evaluate the energy based on the
    //       bar's two nodes' displacement
    double
    computeEnergy(const NodalDisplacements & displacements) const {

      ignoreUnusedVariables(displacements);

      double energyDensity  = 0.0;

      // TODO: Evaluate the energyDensity
      // NOTE: The first input parameter is displacement, not displacement gradient...
      Strain strain = computeBarStrain(displacements);
      energyDensity = _materialModel->computeEnergy(strain);


      // TODO: Based on the energy density, the bar's area and the bar's undeformed
      //       length, evaluate the total energy stored

      double energy = energyDensity * _properties._area * _undeformedBarLength;

      return energy;
    }



    // TODO: Complete the function computeForces to evaluate the forces at all NumberOfNodes
    //       nodes based on the bar's two nodes' displacement
    Forces
    computeForces(const NodalDisplacements & displacements) const {

      //ignoreUnusedVariables(displacements);


      // TODO: Based on displacement (again, be reminded that this is not displacement
      //       gradient!), evaluate the stress tensor
      Strain strain = computeBarStrain(displacements);
      Stress stress = Stress::Zero();
      stress = _materialModel->computeStress(strain);
      Vector                        _deformedBarUnitVector;
      double                        _deformedBarLength;
      _deformedBarLength = ((_X1+displacements[1])-(_X0+displacements[0])).norm();
      _deformedBarUnitVector = ((_X1+displacements[1])-(_X0+displacements[0]))/_deformedBarLength;



      // TODO: Evaluate the forces at all NumberOfNodes nodes

      Forces forces;
      forces[0]= Vector::Zero();
      forces[1]= Vector::Zero();
      forces[0] = - stress(0,0) *_properties._area * _deformedBarUnitVector;
      forces[1] = + stress(0,0) *_properties._area * _deformedBarUnitVector;


      // Return
      return forces;
    }


  private:

    // Private members
    const MaterialModel * _materialModel;

    // TODO: in case you might want to store more things - go for it!

    // ...

  };

  }

  #endif //ELEMENT_TWO_NODE_BAR

```