```cpp
#include "/src/Definitions.h"
#include "/src/ElementTests.h"

#include "MaterialModelBar1D.h"
#include "TwoNodeBar.h"
#include "ExternalForces.h"

// Typedef based on MaterialModel1DBar
typedef MaterialModels::MaterialModel1DBar    MaterialModel;

// Typedef's based on FiniteBar3D
typedef Elements::FiniteBar3D<MaterialModel>  Element;
typedef Elements::Properties                  ElementProperties;
typedef Element::Node                         Node;
typedef Element::Vector                       Vector;
typedef Element::Stress                       Stress;
typedef Element::Strain                       Strain;

// Typedef based on the ConstantBodyForce-element
typedef Elements::ExternalForce::ConstantBodyForce<Element> ConstantBodyForce;


int main(int arc, char *argv[]) {

  ignoreUnusedVariables(arc,argv);

  // Preliminary - Opening the output file. You can obviously change all this
  //               if you feel adventurous
  FILE * file_output_totalEnergy;
  file_output_totalEnergy = fopen("Output_TotalEnergy.csv","w");
  fprintf(file_output_totalEnergy,"VerticalDisplacement, Energy\n");

  FILE * file_output_nodalForce;
  file_output_nodalForce = fopen("Output_NodalForce.csv","w");
  fprintf(file_output_nodalForce,"VerticalDisplacement, HorizontalForceInternal,
  HorizontalForceExternal, VerticalForceInternal, VerticalForceExternal\n");

  // Initalize the 1D Bar Material Model with some chosen constant
  const double youngsModulus = 1.5e3;
  MaterialModel materialModel(youngsModulus);

  // Initalize element properties
  const double area = 1.;
  const double density = 1.;
  ElementProperties elementProperties(area,density);



  // TODO: Initialize the node positions of all three nodes
  const double barlengthUndeformed = 1.0;
  const double phi                 = 3.1415926536/4.0;

  array<Node, 3>   nodes;
  nodes[0]._id = 0;
  nodes[0]._position = {0,0,0};
  nodes[1]._id = 1;
  nodes[1]._position = {sin(phi)*barlengthUndeformed,sin(phi)*barlengthUndeformed,0};
  nodes[2]._id = 2;
  nodes[2]._position = {2*sin(phi)*barlengthUndeformed,0,0};
  //Vector node0 = Vector::Zero();
  //node0 = {0,0,0};
  //Vector node1 = Vector::Zero();
  //node1 ={sin(phi)*barlengthUndeformed,sin(phi)*barlengthUndeformed,0};
  //Vector node2 = Vector::Zero();
  //node2 = {2*sin(phi)*barlengthUndeformed,0,0};



  // TODO: Initialize the two bars based on the nodal locations you've just defined
  // REMINDER: Node is a class whose constructor expects an ID alongside a position
  array<Node,2> nodesComprisingBar;

  // Left Bar
```

```
73      nodesComprisingBar[0] = Node(nodes[0]._id,nodes[0]._position); // ...
74      nodesComprisingBar[1] = Node(nodes[1]._id,nodes[1]._position); // ...
75      //nodesComprisingBar[0] = Node(0,node0); // ...
76      //nodesComprisingBar[1] = Node(1,node1); // ..
77      Element barLeft(nodesComprisingBar, elementProperties, &materialModel);
78
79      // Right Bar
80
81      nodesComprisingBar[0] = Node(nodes[1]._id,nodes[1]._position); // ...
82      nodesComprisingBar[1] = Node(nodes[2]._id,nodes[2]._position);
83      //nodesComprisingBar[0] = Node(1,node1); // ...
84      //nodesComprisingBar[1] = Node(2,node2); // ...
85      Element barRight(nodesComprisingBar, elementProperties, &materialModel);
86
87
88      // TODO: Initialize gravity vector
89      Vector gravityForceVector = Vector::Zero();
90      gravityForceVector(1) = -9.8;// ...
91
92      //ignoreUnusedVariable(gravityForceVector);   // you can delete this very line as
        soon
93                                                  // as you're finished
94
95
96
97      // TODO: Based on the gravity vector, and the two bar elements, create one
98      //       external force element (ConstantBodyForce) per bar element
99
100     // Left Gravity Element
101     ConstantBodyForce gravityElementLeft (barLeft,  gravityForceVector);
102
103     // Right Gravity element
104
105     ConstantBodyForce gravityElementRight (barRight, gravityForceVector);
106
107
108
109     // TODO: 1) Sweep through a range of displacements of the central node as
        illustrated
110     //       on the assignment sheet, evaluate the energy, return it as part of the
111     //       output file and then visualize the energy vs. displacement curve, which -
112     //       except for one discrete exception - should give two minima. You may have to
113     //       play a little bit with your Young's modulus. So long as capture two minima,
114     //       it doesn't necessarily be too physically viable...
115     //       2) Furthermore, return the y-component of the int. force acting on the
        central
116     //       node and show that at the minima, it equates to the forces excerted by the
117     //       ext. force elements
118     array<Vector,2> displacementNodesBarLeft;
119     array<Vector,2> displacementNodesBarRight;
120
121     displacementNodesBarLeft [0] = Vector::Zero();
122     displacementNodesBarLeft [1] = Vector::Zero();
123     displacementNodesBarRight[0] = Vector::Zero();
124     displacementNodesBarRight[1] = Vector::Zero();
125
126     double currentDisplacement     = +1*sin(phi)*barlengthUndeformed;
127     const double deltaDisplacement = 0.005;
128     const double maxDisplacement   = -3*sin(phi)*barlengthUndeformed;
129
130     while (currentDisplacement > maxDisplacement){
131
132       // TODO: Set displacementNodesBarLeft, displacementNodesBarRight for current
          displacement
133       // NOTE: The left node of the left bar as well as the right node of the right
          bar are
134       //       fixed, so based on the above zeroing of all displacments, really, there
          is no
135       //       need to change these two.
136       displacementNodesBarLeft[1](1) = currentDisplacement;
137       displacementNodesBarRight[0](1) = currentDisplacement;
138
139       // TODO: Evaluate total energy comprising contributions from the left and right
```

```
      bar's
      //       stored energy as well as the work performed by gravity
      double energy = barLeft.computeEnergy(displacementNodesBarLeft)
                    + barRight.computeEnergy(displacementNodesBarRight)
                    + gravityElementLeft.computeEnergy(displacementNodesBarLeft)
                    + gravityElementRight.computeEnergy(displacementNodesBarRight);
                    // ...

      fprintf(file_output_totalEnergy,"%6.4f, %8.4f\n",currentDisplacement,energy);



      // TODO: Evaluate total forces
      Element::Forces forceBarLeft
        = barLeft.computeForces(displacementNodesBarLeft);
      Element::Forces forceBarRight
        = barRight.computeForces(displacementNodesBarRight);

      ConstantBodyForce::Forces forceGravityLeft
        = gravityElementLeft.computeForces(displacementNodesBarLeft);
      ConstantBodyForce::Forces forceGravityRight
        = gravityElementRight.computeForces(displacementNodesBarRight);



      fprintf(file_output_nodalForce,"%6.4f, %8.4f, %8.4f, %8.4f, %8.4f\n",
                currentDisplacement                                ,
                forceBarLeft[1](0)    + forceBarRight[0](0)    ,
                forceGravityLeft[1](0)+ forceGravityRight[0](0),
                forceBarLeft[1](1)    + forceBarRight[0](1)    ,
                forceGravityLeft[1](1)+ forceGravityRight[0](1));


      // Incrementally update current displacement
      currentDisplacement -= deltaDisplacement;

    }

    // Close and return
    fclose(file_output_totalEnergy);
    fclose(file_output_nodalForce );

    // Return
    return 0;

  }
```