```cpp
#ifndef ASSEMBLER_H
#define ASSEMBLER_H

#include "Definitions.h"
#include "Utilities.h"

template <class E>
class Assembler {

public:

  typedef E                                                Element;
  typedef typename Element::Vector                         ElementVector;
  typedef typename Element::Forces                         ElementForces;
  typedef typename Element::StiffnessMatrix                ElementStiffnessMatrix;
  typedef typename Element::MassMatrix                     ElementMassMatrix;
  typedef typename Element::Stress                         ElementStress;
  static const unsigned int NumberOfNodesPerElement =      Element::NumberOfNodes;
  static const unsigned int SpatialDimension =             Element::SpatialDimension;
  static const unsigned int DegreesOfFreedom =             Element::DegreesOfFreedom;


  Assembler () :_numberOfNodes(0) {
  }

  Assembler (const size_t numberOfNodes ) :_numberOfNodes(numberOfNodes) {
  }

  Assembler ( const vector<Element> & elements      ,
              const size_t            numberOfNodes ) :
      _elements(elements),
      _numberOfNodes(numberOfNodes) {
  }



  MatrixXd
  assembleConsistentMassMatrix() const {

    size_t numberOfDofs = _numberOfNodes * DegreesOfFreedom;
    Eigen::MatrixXd consistentMassMatrix(numberOfDofs, numberOfDofs);
    consistentMassMatrix.fill(0);

    // TODO: Fill in the consistent mass matrix - as mentioned in the project assignment
    //       there may be a lot of similarities to the assembly of the stiffness matrix
    for (size_t elementIndex = 0; elementIndex < _elements.size(); ++elementIndex) {
      const Element & element = _elements[elementIndex];
      array<size_t, NumberOfNodesPerElement> elementNodeIds =
        element.getNodeIds();
      ElementMassMatrix elementMassMatrix =
        element.computeConsistentMassMatrix();


      for (size_t nodeIndex1 = 0; nodeIndex1 < elementNodeIds.size();
           ++nodeIndex1) {
        size_t nodeId1 = elementNodeIds[nodeIndex1];
        for (size_t nodeIndex2 = 0; nodeIndex2 < elementNodeIds.size();
             ++nodeIndex2) {
          size_t nodeId2 = elementNodeIds[nodeIndex2];
          for (size_t i = 0; i < DegreesOfFreedom; ++i) {
            for (size_t j = 0; j < DegreesOfFreedom; ++j) {
              consistentMassMatrix(nodeId1 * DegreesOfFreedom + i,
                                   nodeId2 * DegreesOfFreedom + j) +=
                elementMassMatrix(nodeIndex1 * DegreesOfFreedom + i,
                                  nodeIndex2 * DegreesOfFreedom + j);
            }
          }
        }
      }
    }
    return stiffnessMatrix;
```

```
72
73
74        }
75
76        return consistentMassMatrix;
77
78      };
79
80
81
82      // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
83      // All the following functions are given, nothing needs to be changed
84      // They're the solution to the last problem set. Feel free to use your
85      // own, if you think you did a better job in implementing everything :)
86      // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
87
88
89      //%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%
90      //%%%%%%%%%%%%%%%%%%%%                    Assemble energy
        %%%%%%%%%%%%%%%%%%%%
91      //%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%
92      double
93      assembleEnergy(const vector<ElementVector> & displacements) const {
94        double energy = 0.0;
95        // normal elements
96        for (size_t elementIndex = 0; elementIndex < _elements.size(); ++elementIndex) {
97          const Element & element = _elements[elementIndex];
98          array<size_t, NumberOfNodesPerElement> elementNodeIds =
99            element.getNodeIds();
100         array<ElementVector, NumberOfNodesPerElement> elementDisplacements =
101           Utilities::getElementDisplacementsFromGlobalList<Element>(elementNodeIds,
102                                                                      displacements);
103         energy += element.computeEnergy(elementDisplacements);
104       }
105       return energy;
106     }
107
108
109
110
111     //%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%
112     //%%%%%%%%%%%%%%%%%%%%                    Assemble forces
        %%%%%%%%%%%%%%%%%%%%
113     //%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%
114
115     Eigen::VectorXd
116     assembleForceVector(const vector<ElementVector> & displacements) const {
117       size_t numberOfDofs = displacements.size() * DegreesOfFreedom;
118       Eigen::VectorXd forceVector(numberOfDofs);
119       forceVector.fill(0);
120
121       // normal elements
122       for (size_t elementIndex = 0; elementIndex < _elements.size(); ++elementIndex) {
123         const Element & element = _elements[elementIndex];
124         array<size_t, NumberOfNodesPerElement> elementNodeIds =
125           element.getNodeIds();
126         array<ElementVector, NumberOfNodesPerElement> elementDisplacements =
127           Utilities::getElementDisplacementsFromGlobalList<Element>(elementNodeIds,
128                                                                      displacements);
129         ElementForces elementForces = element.computeForces(elementDisplacements);
130         for (size_t nodeIndex = 0; nodeIndex < elementNodeIds.size();
131              ++nodeIndex) {
132           size_t nodeId = elementNodeIds[nodeIndex];
133           for (size_t i = 0; i < DegreesOfFreedom; ++i) {
134             forceVector(nodeId * DegreesOfFreedom + i) +=
135               elementForces[nodeIndex](i);
136           }
137         }
138       }
```

```
    return forceVector;
  }





  //%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%
  //%%%%%%%%%%%%%%%%%%%%               Assemble stiffness
  %%%%%%%%%%%%%%%%%%%%
  //%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%

  Eigen::MatrixXd
  assembleStiffnessMatrix(const vector<ElementVector> & displacements) const {
    size_t numberOfDofs = displacements.size() * DegreesOfFreedom;
    Eigen::MatrixXd stiffnessMatrix(numberOfDofs, numberOfDofs);
    stiffnessMatrix.fill(0);

    // normal elements
    for (size_t elementIndex = 0; elementIndex < _elements.size();
         ++elementIndex) {
      const Element & element = _elements[elementIndex];
      array<size_t, NumberOfNodesPerElement> elementNodeIds =
        element.getNodeIds();
      array<ElementVector, NumberOfNodesPerElement> elementDisplacements =
        Utilities::getElementDisplacementsFromGlobalList<Element>(elementNodeIds,
                                                                  displacements);
      ElementStiffnessMatrix elementStiffnessMatrix =
        element.computeStiffnessMatrix(elementDisplacements);
      for (size_t nodeIndex1 = 0; nodeIndex1 < elementNodeIds.size();
           ++nodeIndex1) {
        size_t nodeId1 = elementNodeIds[nodeIndex1];
        for (size_t nodeIndex2 = 0; nodeIndex2 < elementNodeIds.size();
             ++nodeIndex2) {
          size_t nodeId2 = elementNodeIds[nodeIndex2];
          for (size_t i = 0; i < DegreesOfFreedom; ++i) {
            for (size_t j = 0; j < DegreesOfFreedom; ++j) {
              // heaven help me if this is wrong
              stiffnessMatrix(nodeId1 * DegreesOfFreedom + i,
                              nodeId2 * DegreesOfFreedom + j) +=
                elementStiffnessMatrix(nodeIndex1 * DegreesOfFreedom + i,
                                       nodeIndex2 * DegreesOfFreedom + j);
            }
          }
        }
      }
    }
    return stiffnessMatrix;
  };



  //%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%
  //%%%%%%%%%%%%%%%%%%%%             Compute nodal stresses
  %%%%%%%%%%%%%%%%%%%%
  //%%%%%%%%%%%%%%%%%%%%
  %%%%%%%%%%%%%%%%%%%%

  vector<ElementStress>
  computeNodalStresses(const vector<ElementVector> & displacements) const {

    const size_t numberOfNodes = displacements.size();

    vector<ElementStress> nodalStresses(numberOfNodes, ElementStress::Zero());
    vector<double> volumeSums(numberOfNodes, 0);

    vector<ElementStress> elementStresses = computeElementStresses(displacements);

    for (size_t elementIndex = 0; elementIndex < _elements.size(); ++elementIndex) {
```

```
206         const Element & element = _elements[elementIndex];
207         array<size_t, NumberOfNodesPerElement> elementNodeIds = element.getNodeIds();
208         array<double, NumberOfNodesPerElement> elementWeights =
209           element.computeNodalWeights();
210         const ElementStress & elementStress = elementStresses[elementIndex];
211
212         for (size_t nodeIndex = 0; nodeIndex < elementNodeIds.size(); nodeIndex++) {
213           nodalStresses[elementNodeIds[nodeIndex]] += elementStress *
              elementWeights[nodeIndex];
214           volumeSums[elementNodeIds[nodeIndex]] += elementWeights[nodeIndex];
215         }
216       }
217
218       for (size_t nodeIndex = 0; nodeIndex < numberOfNodes; nodeIndex++) {
219         nodalStresses[nodeIndex] /= volumeSums[nodeIndex];
220       }
221       return nodalStresses;
222     }
223
224
225
226
227     //%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%
228     //%%%%%%%%%%%%%%%%%%%%              Compute nodal strains
      %%%%%%%%%%%%%%%%%%%%
229     //%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%
230
231     vector<typename Element::Strain>
232     computeNodalStrains(const vector<typename Element::Vector> & displacements) const {
233
234       const size_t numberOfNodes = displacements.size();
235
236       vector<typename Element::Strain> nodalStrains(numberOfNodes,
237                                                     Element::Strain::Zero());
238       vector<double> volumeSums(numberOfNodes, 0);
239
240       vector<typename Element::Strain> elementStrains =
241         computeElementStrains(displacements);
242
243       for (size_t elementIndex = 0; elementIndex < _elements.size(); ++elementIndex) {
244         const Element & element = _elements[elementIndex];
245         array<size_t, Element::NumberOfNodes> elementNodeIds = element.getNodeIds();
246         array<double, Element::NumberOfNodes> elementWeights =
247           element.computeNodalWeights();
248         const typename Element::Strain & elementStrain = elementStrains[elementIndex];
249
250         for (size_t nodeIndex = 0; nodeIndex < elementNodeIds.size(); nodeIndex++) {
251           nodalStrains[elementNodeIds[nodeIndex]] +=
252             elementStrain / elementWeights[nodeIndex];
253           volumeSums[elementNodeIds[nodeIndex]] += 1./elementWeights[nodeIndex];
254         }
255       }
256
257       for (size_t nodeIndex = 0; nodeIndex < numberOfNodes; nodeIndex++){
258         nodalStrains[nodeIndex] /= volumeSums[nodeIndex];
259       }
260       return nodalStrains;
261     }
262
263
264
265
266
267     //%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%
268     //%%%%%%%%%%%%%%%%%%%%            Compute element stresses
      %%%%%%%%%%%%%%%%%%%%
269     //%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%
270
271     vector<ElementStress>
```

```
272     computeElementStresses(const vector<ElementVector> & displacements) const {
273
274       vector<ElementStress> allElementStresses(_elements.size());
275       for (size_t elementIndex = 0; elementIndex < _elements.size(); ++elementIndex) {
276         const Element & element = _elements[elementIndex];
277         array<size_t, NumberOfNodesPerElement> elementNodeIds = element.getNodeIds();
278         array<ElementVector, NumberOfNodesPerElement> elementDisplacements =
279
            Utilities::getElementDisplacementsFromGlobalList<Element>(elementNodeIds,displ
            acements);
280         array<ElementStress, Element::QuadPoints> elementStresses =
281           element.computeStressesAtGaussPoints(elementDisplacements);
282         ElementStress average = ElementStress::Zero();
283         for (size_t qpIndex = 0; qpIndex < Element::QuadPoints; ++qpIndex) {
284           average += elementStresses[qpIndex];
285         }
286         average /= Element::QuadPoints;
287         allElementStresses[elementIndex] = average;
288       }
289       return allElementStresses;
290     }
291
292
293
294
295
296     //%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%
297     //%%%%%%%%%%%%%%%%%%%%            Compute element strains
      %%%%%%%%%%%%%%%%%%%%
298     //%%%%%%%%%%%%%%%%%%%%
      %%%%%%%%%%%%%%%%%%%%
299
300     vector<typename Element::Strain>
301     computeElementStrains(const vector<typename Element::Vector> & displacements)
      const {
302
303       vector<typename Element::Strain> allElementStrains(_elements.size());
304       for (size_t elementIndex = 0; elementIndex < _elements.size(); ++elementIndex) {
305         const Element & element = _elements[elementIndex];
306         array<size_t, Element::NumberOfNodes> elementNodeIds = element.getNodeIds();
307         array<typename Element::Vector, Element::NumberOfNodes> elementDisplacements =
308           ::Utilities::getElementDisplacementsFromGlobalList<Element>(elementNodeIds,
309                                                                       displacements);
310         array<typename Element::Strain, Element::QuadPoints> elementStrains =
311           element.computeStrainsAtGaussPoints(elementDisplacements);
312         typename Element::Strain average = Element::Strain::Zero();
313         for (size_t qpIndex = 0; qpIndex < Element::QuadPoints; ++qpIndex) {
314           average += elementStrains[qpIndex];
315         }
316         average /= Element::QuadPoints;
317         allElementStrains[elementIndex] = average;
318       }
319       return allElementStrains;
320     }
321
322
323
324
325     //%%%%%%%%%%%%%%%%%%%%                                        %%%%%%%%%%%%%%%%%%%%
326     //%%%%%%%%%%%%%%%%%%%%            Get-Functions               %%%%%%%%%%%%%%%%%%%%
327     //%%%%%%%%%%%%%%%%%%%%                                        %%%%%%%%%%%%%%%%%%%%
328
329     size_t
330     getNumberOfElements() const {
331       return _elements.size();
332     }
333
334     size_t
335     getNumberOfNodes() const {
336       return _numberOfNodes;
337     }
338
```

```
339   private:
340
341     vector<Element> _elements;
342     const size_t _numberOfNodes;
343   };
344
345   #endif  // ASSEMBLER_H
346
```