```cpp
#include "mpi.h"
#include "Definitions.h"
#include "MeshUtilities.h"
#include "Quadrature.h"
#include "PostProcessorVtk.h"
#include "ElementTests.h"

#include "MaterialModelNeoHookean.h"
#include "ElementTypes.h"
#include "Wall.h"

#include "IsoparametricElement.h"
#include "Assembler.h"
#include "SolverImplicit.h"

const unsigned int         SpatialDimension = 3;
const unsigned int         DegreesOfFreedom = 3;
const unsigned int numberOfQuadraturePoints = 1;

typedef MaterialModels::NeoHookean<SpatialDimension>                MaterialModel;

typedef ElementTypes::Simplex<SpatialDimension>                     ElementType;
typedef Elements::IsoparametricElement<MaterialModel,
                                       numberOfQuadraturePoints,
                                       ElementType,
                                       DegreesOfFreedom>            Element;
typedef Element::Properties                                         ElementProperties;
typedef Element::Node                                               Node;
typedef Element::Vector                                             Vector;
typedef Element::Point                                              Point;
typedef Element::Stress                                             Stress;
typedef Element::Strain                                             Strain;

typedef SingleElementMesh<Element>                                  Mesh;

typedef Element                                                     PhysicalElement;
typedef Elements::ExternalForce::Wall<SpatialDimension,DegreesOfFreedom>
                                                                    ExternalElement;
typedef Assembler<PhysicalElement>                                  PhysicalAssembler;
typedef Assembler<ExternalElement>                                  ExternalAssembler;
typedef SolverImplicitDynamics<PhysicalAssembler,ExternalAssembler
                                                ,PhysicalAssembler> Solver;


const unsigned int NumberOfNodesPerElement = Element::NumberOfNodes;


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

  ignoreUnusedVariables(arc,argv);

    // The following lines simply create an output directory
  char sprintfBuffer[500];
  sprintf(sprintfBuffer,"Output_Main4");
  const string outputPath = string(sprintfBuffer);

  printf("Writing files to %s\n", outputPath.c_str());
  const bool createNewDirectories = true;
  Utilities::directoryCreator(outputPath, createNewDirectories, Quiet);

  // %%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%
  // %%%%%%%%%%%%%%%%%% Problem 4) (i) Creation of material model %%%%%%%%%%%%%%%%%%
  // %%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%

  // TODO: Create your materialModel

  // ...
  const double materialModelParameter0 = 5.0 * power(10,5);
  const double materialModelParameter1 = 5.0 * power(10,7);

  MaterialModel materialModel(materialModelParameter0,materialModelParameter1);

  // %%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%
```

```cpp
    // %%%%%%%%%%%%%%%%%% Problem 4) (ii) Creation of mesh           %%%%%%%%%%%%%%%%%%
    // %%%%%%%%%%%%%%%%%%                                              %%%%%%%%%%%%%%%%%%

    Mesh mesh;
    MeshUtilities::readMeshFromFile("SphereTetMesh_002000.dat",&mesh);

    const size_t numberOfNodes    = mesh._nodes.size();
    const size_t numberOfElements = mesh._connectivity.size();


    //%%%%%%%%%%%%%%%%%%%%                                            %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%% Problem 4) (iii) Preliminary stuff for elements %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%%                                            %%%%%%%%%%%%%%%%%%%%

    // Create the element type and the element properties
    const double elementMultiplier = 1522;
    ElementType elementType;
    ElementProperties elementProperties(elementMultiplier);

    // Choose a quadrature rule
    const QuadratureRule<Element::SpatialDimension, numberOfQuadraturePoints>
    quadratureRule = Quadrature::buildSimplicialQuadrature<Element::SpatialDimension, numberOfQuadraturePoints>();

    ignoreUnusedVariables(elementType);
    ignoreUnusedVariables(quadratureRule);

    //%%%%%%%%%%%%%%%%%%%%                                            %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%% Problem 4) (iv) Creation of the elements %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%%                                            %%%%%%%%%%%%%%%%%%%%

    // Wall parameter
    const double wallStrength = 1.0*1.0e8;
    Vector wallOriginPosition = Vector::Zero(); wallOriginPosition (2)= +0.0;
    Vector wallNormalDirection= Vector::Zero(); wallNormalDirection(2)= -1.0;

    ignoreUnusedVariables(wallStrength);

    // TODO: Collect all external elements
    vector<ExternalElement> externalElements; externalElements.clear();
    for (unsigned int indexNode = 0; indexNode < numberOfElements /* TODO: set */; indexNode++){
      // ...
       // Read out the nodes corresponding no the indexElement'th element
          array<Node,SpatialDimension+1> nodesSimplex;
          for (unsigned int indexNode = 0; indexNode < SpatialDimension+1; indexNode++)
      {
              nodesSimplex[indexNode] = mesh._nodes[mesh._connectivity[indexElement][indexNode]];
          }
      Element simplexElement(nodesSimplex,
                             elementProperties,
                             elementType,
                           & quadratureRule,
                           & materialModel);
      // REMINDER: You can push new elements into a vector via the .push_back option
      externalElements.push_back(simplexElement);
    }


    // TODO:Collect all physical elements
    vector<PhysicalElement> physicalElements; physicalElements.clear();
    for (unsigned int indexElement = 0; indexElement < numberOfElements /* TODO: set */; indexElement++){
        // ...
           // Read out the nodes corresponding no the indexElement'th element
          array<Node,SpatialDimension+1> nodesSimplex;
          for (unsigned int indexNode = 0; indexNode < SpatialDimension+1; indexNode++)
      {
              nodesSimplex[indexNode] =
```

```
                mesh._nodes[mesh._connectivity[indexElement][indexNode]];
        }
      Element simplexElement(nodesSimplex,
                             elementProperties,
                             elementType,
                           & quadratureRule,
                           & materialModel);
      // REMINDER: You can push new elements into a vector via the .push_back option
      physicalElements.push_back(simplexElement);

      }


    //%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%% Problem 4) (v) Creation of an assembler %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%

    // TODO: Create assemblers corresponding to your physical and external elements

    // ...
      PhysicalAssembler physicalAssembler(physicalElements, numberOfNodes);
      ExternalAssembler externalAssembler(externalElements, numberOfNodes);

    //%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%% Problem 4) (vi) Creation of solver %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%%                                                %%%%%%%%%%%%%%%%%%%%

    // TODO: Create an object of your SolverImplicitDynamics class

    // ...

    Solver solver(physicalAssembler,externalAssembler,physicalAssembler);

    const unsigned int  maxIterations     = 1000 ;
    const double        tolerance         = 1e-4  ;



    //%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%% Problem 4) (vii) Initialisation %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%

    // TODO: Initiate all states that you need for your solver
    vector<Vector> currentNodalDisplacement(numberOfNodes,Vector::Zero());
    // ...
    vector<Vector> currentNodalAcceleration(numberOfNodes,Vector::Zero());
    vector<Vector> currentNodalVelocity(numberOfNodes,Vector::Vector::Zero());
    // TODO: Impose the initial velocity
    // ...
    for (unsigned int nodeIndex = 0; nodeIndex < numberOfNodes; nodeIndex++) {
        for (unsigned int dofIndex = 0; dofIndex < DegreesOfFreedom; dofIndex++) {
          if (dofIndex == 0){
              currentNodalVelocity[nodeIndex](dofIndex) = 0;
          }
          else if (dofIndex == 1){
              currentNodalVelocity[nodeIndex](dofIndex) = 0;
          }
          else if (dofIndex == 2){
              currentNodalVelocity[nodeIndex](dofIndex) = -10;
          }
        }
      }

    // Empty boundary conditions
    vector<EssentialBoundaryCondition> emptyEssentialBoundaryConditions;
    emptyEssentialBoundaryConditions.clear();

    //%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%% Problem 4) (viii) Run %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%%                                %%%%%%%%%%%%%%%%%%%%

```

```
210     // TODO: Chose the number of loadsteps
211     const unsigned int numberOfLoadsteps = 100; // ...
212
213     for (unsigned int loadstepIndex = 0; loadstepIndex < numberOfLoadsteps;
        loadstepIndex++){
214
215       if (loadstepIndex % unsigned(1) == 0) {
216         printf("\ntimestep %6u (%%%5.1f) at %s\n",
217                  loadstepIndex,
218                  100. * loadstepIndex / float(numberOfLoadsteps),
219                  Utilities::getLocalTimeString().c_str());
220       }
221
222       // TODO: Call solver
223       // ...
224
225       vector<Vector> displacements
226          = solver.computeNewmarkUpdate(essentialBCs,
           currentNodalDisplacement,currentNodalVelocity,currentNodalAcceleration,
           maxIterations, tolerance, true);
227
228       if (!(loadstepIndex%1)){
229
230         printf("Giving output at loadstep (%d/%d).\n",loadstepIndex,numberOfLoadsteps);
231
232         // TODO: define the following four vectors
233         const vector<Stress> nodeStresses    (numberOfNodes   ,Stress::Zero()); // ...
234         const vector<Stress> elementStresses (numberOfElements,Stress::Zero()); // ...
235         const vector<Strain> nodeStrains     (numberOfNodes   ,Strain::Zero()); // ...
236         const vector<Strain> elementStrains  (numberOfElements,Stress::Zero()); // ...
237           nodeStresses = assembler.computeElementStresses(displacements);
238           elementStresses = assembler.computeNodalStresses (displacements) ;
239           nodeStrains = assembler.computeElementStrains(displacements);
240           elementStrains= assembler.computeNodalStrains(displacements);
241
242
243         // !!! NOTHING TO BE DONE FROM HERE ONWARDS
            !!!
244         sprintf(sprintfBuffer,"%s/WallImpact_%03u",outputPath.c_str(),loadstepIndex);
245
246         PostProcessors::Vtk::NamedArrays<int,double> vtkNamedArrays;
247         PostProcessors::Vtk::makeDeformedMeshFile<Element>( mesh,
248                                                             currentNodalDisplacement,
249                                                             nodeStresses,
250                                                             elementStresses,
251
                                                            MaterialModel::getStressComp
                                                            onentNames(),
252                                                             nodeStrains,
253                                                             elementStrains,
254
                                                            MaterialModel::getStrainComp
                                                            onentNames() ,
255
                                                            emptyEssentialBoundaryCondit
                                                            ions,
256                                                             string(sprintfBuffer),
257                                                             vtkNamedArrays);
258       }
259
260     }
261
262     return 0;
263   }
264
```