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

#include "MaterialModelBar1D.h"
#include "Wall.h"

#include "TwoNodeBar.h"
#include "Assembler.h"
#include "SolverImplicit.h"

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

typedef MaterialModels::MaterialModel1DBar                          MaterialModel;
typedef Elements::FiniteBar<MaterialModel,SpatialDimension>         Element;
typedef Elements::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_Main6");
  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

  // ...
  MaterialModel materialModel (youngModulus);


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

  Mesh mesh;

  const string meshFileName = "crossUnitCube.dat";
  MeshUtilities::readMeshFromFile<Element>(meshFileName,&mesh);

  array<size_t, SpatialDimension> numberOfCubesPerSide = {{5,5,5}};
  const double preperiodicSpatialTolerance = 1e-4;
  const double sideLength = 1.0;

  Vector unitXVector = Vector::Zero(); unitXVector(0) = sideLength;
  Vector unitYVector = Vector::Zero(); unitYVector(1) = sideLength;
  Vector unitZVector = Vector::Zero(); unitZVector(2) = sideLength;

  const array<Vector,SpatialDimension> patternVectors
    = {{unitXVector,unitYVector,unitZVector}};

  MeshUtilities::buildPeriodicMeshFromUnitCell(patternVectors             ,
                                               numberOfCubesPerSide       ,
                                               &mesh                      ,
                                               preperiodicSpatialTolerance);

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

  cout << "Number of nodes in the mesh: "               << numberOfNodes    << endl;
  cout << "Number of beam elements in the mesh: "       << numberOfElements << endl;


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

  // Create the element type and the element properties
  const double barDensity = 1522;
  const double barArea    = 1.0 ;
  ElementProperties elementProperties(barArea,barDensity);


  //%%%%%%%%%%%%%%%%%%%%                                          %%%%%%%%%%%%%%%%%%%%
  //%%%%%%%%%%%%%%%%%%%% 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);
  }


  // 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

    // ...


    //%%%%%%%%%%%%%%%%%%%%                          %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%% Problem 4) (vi) Solver   %%%%%%%%%%%%%%%%%%%%
    //%%%%%%%%%%%%%%%%%%%%                          %%%%%%%%%%%%%%%%%%%%

    // TODO: Create an object of your SolverImplicitDynamics class

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

    // TODO: Chose the number of loadsteps
```

```
  const unsigned int numberOfLoadsteps = 100; // ...

  for (unsigned int loadstepIndex = 0; loadstepIndex < numberOfLoadsteps;
  loadstepIndex++){

    if (loadstepIndex % unsigned(1) == 0) {
      printf("\ntimestep %6u (%%%5.1f) at %s\n",
               loadstepIndex,
               100. * loadstepIndex / float(numberOfLoadsteps),
               Utilities::getLocalTimeString().c_str());
    }

    // TODO: Call solver
    // ...
    vector<Vector> displacements
            = solver.computeNewmarkUpdate(essentialBCs,
            currentNodalDisplacement,currentNodalVelocity,currentNodalAcceleration,
            maxIterations, tolerance, true);

    if (!(loadstepIndex%1)){

      printf("Giving output at loadstep (%d/%d).\n",loadstepIndex,numberOfLoadsteps);

      // TODO: set elementStresses
      const vector<Stress> elementStresses (numberOfElements,Stress::Zero()); // ...
      elementStresses = assembler.computeNodalStresses (displacements) ;

      // Paraview Output
      PostProcessors::Vtk::NamedArray<double> vtkStresses;
      vtkStresses._title="Bar Stresses";
      vtkStresses._elementWiseOrNodeWise = PostProcessors::Vtk::ElementWise;

      for (unsigned int elementIndex =0 ; elementIndex <numberOfElements;
      elementIndex++ )
      {
        vtkStresses._array.push_back(elementStresses[elementIndex](0));
      }

      PostProcessors::Vtk::NamedArrays<int,double> vtkNamedArrays;
      vtkNamedArrays.addArray(vtkStresses);


      char
      outputFileDesignation[500];

      sprintf(outputFileDesignation,"%s/WallImpactTruss_%03u",outputPath.c_str(),loads
      tepIndex);

      PostProcessors::Vtk::makeDeformedMeshFile<Element>(
      mesh                            ,
                                                          currentNodalDisplacement
                                                              ,
                                                          emptyEssentialBoundaryCondit
                                                          ions,
                                                          string(outputFileDesignation
                                                          )   ,
                                                          vtkNamedArrays
                                                              );
    }

  }

  return 0;
}


```