UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Barcelona, 14.05.2018

Implementation of a FEM code

Homework 2 part a

PROGRAMMING FOR ENGINEERS AND SCIENTISTS
2017/18

Professor	Amir Abdollahi
Students	Marin Enrico
	Pan Lei
	Rubino Marcello

SUMMARY

1. Introduction

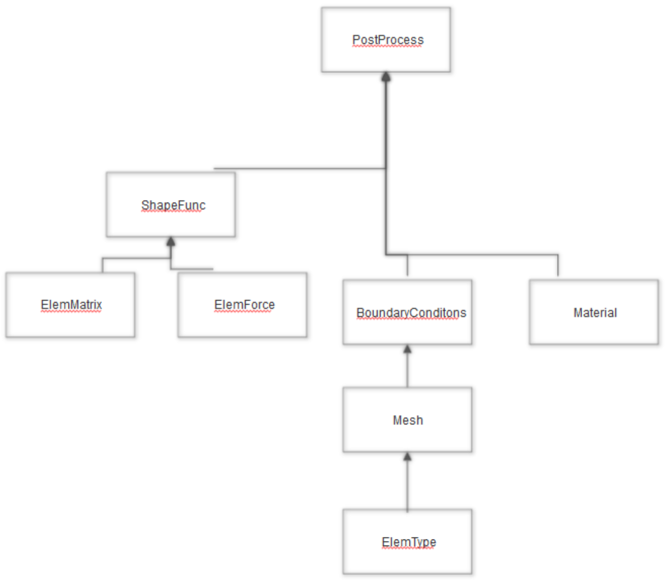
The purpose of this work is to design an initial scheme of a FEM code in C++ to solve a typical fluid dynamic problem, the definition of the velocity field described by the Poisson equation in 2D/3D dimension.
If the flow is irrotational, the velocity can be expressed in terms of a velocity potential. The velocity field, therefore, is constructed from the velocity generated by a scalar potential *u* generated by the mass sources. So, we have developed a C++ program to solve the following possible Poisson equation:

$$\begin{aligned} \nabla \cdot (k(x,y)\nabla u) = 0 \qquad & in \;\; \Omega \\ \nabla u \cdot \boldsymbol{n} = g \qquad & on \;\; \Gamma_N \\ u = u_0 \qquad & on \;\; \partial\Omega \setminus \Gamma_N \end{aligned}$$

where Ω is the computational domain, $\partial\Omega$ is the boundary, $\boldsymbol{n}$ is the outward unit normal vector and $k(x,y)$ is a not constant material property.
In our program, we are going to consider two types of elements: triangular and tetrahedral element.

2. Entity Relationship Diagram

Entity Relationship diagrams are commonly used to explain the relationships among classes in the design. Classes are shown as boxes around the class name:



3. Classes Interface Definitions

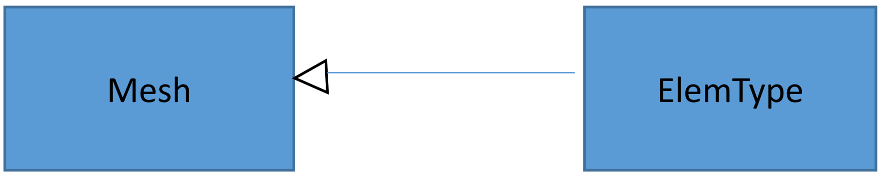
We implemented inheritance relationships in the object design declaring by the use of private, public, protected members. A subclass inherits from one or more superclasses, the private members of the superclass are hidden from the subclass. Only the public and protected members are available to the subclass. Declaring a member private preserves encapsulation, but limits the usefulness of future subclasses.
In the next part of the report, we describe all the header files containing Parent classes and Child classes that we should implement in the C++ FEM code:

3.1 Mesh

Mesh.h	

class Mesh {
        Public: Int Tmesh(); Real Xmesh();
        Private: Int T; Real X;
}
 	
Inline Int MeshTopology::Tmesh()
 	
Inline Real MeshTopology::Xmesh()
 	
//read and storage T,X matrices to
// discretize the domain
	class ElemType : public MeshTopology {
	        Public: Int elemtype(); Int totelem(); Int totnodes();
	        Private: Int elem_type; Int tot_elem; Int tot_nodes;
	
	}
	
	Inline Int ElemType :: elemtype()
	
	Inline Int ElemType :: totelem()
	
	Inline Int ElemType :: totnodes()
	
	//it builds each elements of the mesh

The class ElemType is the derived class of class Mesh.



Mesh class is used for creating the nodes coordinates from the matrix X and the connectivity matrix T. The private data includes array T and X.
ElemType class takes charge of getting the elements type, like using triangular or quadrilateral elements for the problem, the total numbers of the nodes and the elements.
The private data are elem_type, tot_elem and tot_nodes.

3.2 Boundary Conditions

BoundaryConditions.h

```
class Bcondition {
            Public: Real Neu_BC(); Real Dir_BC();
            Private: Real Neu_BC; Real Dir_BC;
}
Inline Real Bcondition :: Neu_BC()
Inline Real Bcondition :: Dir_BC()

//implement the Neumann and Dirichelet BCs
// reading and storing edges and nodes
```

The scope of the class Bcondition is implementing the Neumann and Dirichelet boundary conditions. The private data here are array Neu_BC and array Dir_BC.

3.3 Element

element.h

```
class Shapefunc {
        Public: Real: N(); Nξ (); Real Nη(); N ζ(); Jacob(); invJacob(); defJacob(); Nx(); Ny(); Nz();
        Protected: Real GaussPoints (w_gi; z_gi)
}
Inline Real shapefunc :: N()
…
//consider 3D, 2D, 1D problems
```

```
class ElemMatrix : public shapefunc {
        Public: Real:  K(); M(); C();
}
Inline Int elemMatrix ::K()
…
// K= stiffness matrix
// M = mass matrix
// C= convective matrix
// Compute with different functions elemental Matrices
```

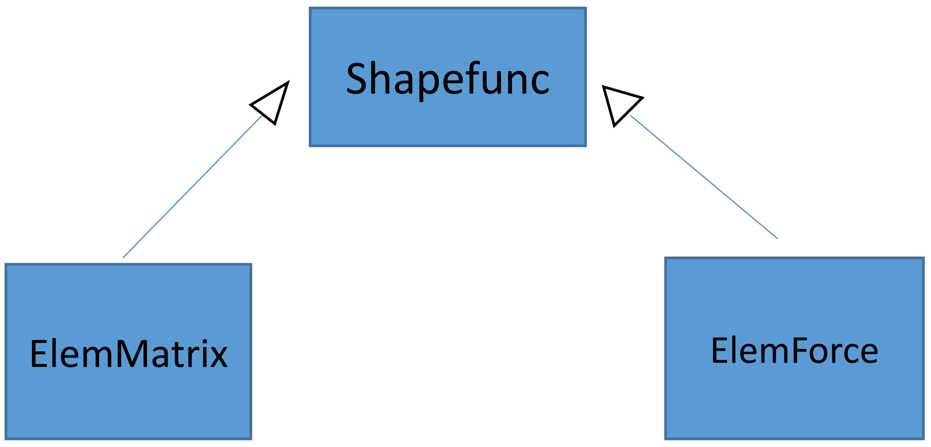
```
class ElemForce : public shapefunc {
        Public:  Real f();

}

Inline Int ElemForce :: f()

// Compute rhs using shapefunctions
```

The class ElemMatrix and class ElemForce have inherited from the class Shapefunc.



The class Shapefunc is intended to deal with the different shape functions for 1D, 2D and 3D problems. As we know, the shape functions in different dimensions varies.
The inline functions include: N(), Nξ (), Nη(), N ζ(), Jacob(), invJacob(), defJacob(), Nx(), Ny() and Nz().
The data here is the array GaussPoints (wgi; zgi) and it is protected type.

The class eEemMatrix is derived from the class Shapefunc and the inherit type is public. And it has added inline functions K, M and C for computing stiffness matrix, mass matrix and convective matrix.
The class ElemForce is also the public inheritance of class shapefunc. Despite the functions inherited from the class shapefunc, class ElemForce has added the inline function f for computing the right hand side of the equation system.

3.4 Material

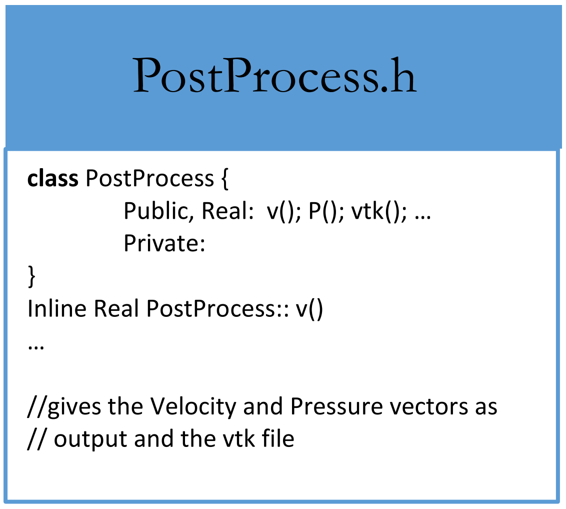
Material.h

```
class Material {
            Public, Real:  k(); E(); v(); T(); …
            Private:
}
Inline Real Material :: k()
…

// it considers all the possible problems such as elastic, termic,
mechanical, fluid problems.
// it implements problems in 3D, 2D, 1D dimensions.
```

The material properties have been considered by the class Material. Inline functions include functions that deal with the properties for different problems like fluid or solid problems.

3.5 PostProcess


PostProcess.h

```
class PostProcess {
            Public, Real:  v(); P(); vtk(); …
            Private:
}
Inline Real PostProcess:: v()
…

//gives the Velocity and Pressure vectors as
// output and the vtk file
```

The class PostProcess is designed for handling the computing results of velocity and pressure. This class can give us the computing results through the special file vtk so that we are able to use the results to get the plots which we are interested in.

4. Properties and alternatives

4.1 Good properties

The OOP programming encourages us to use objects instead of structured programming and focusing on the functions. The object has combined the data and the operators, making sure the safety of the data. And another advantage of OOP is that it can make the coding easier because the classes are convenient to be transplanted and extended by inheritance. In our program, we have built 8 classes which have fully exploited the features of class and every one of them has clear functionality. Each class that we create can be easily used for other FEM applications, which is not limited in just this problem.

4.2 Drawbacks

The use of OOP definitely improves the safety of the data and the classes can be easily transplanted to other FEM program. However, when we achieve the safety of the data by using C++, we also lose the speed of computing compared to Fortran, which is very import in the scientific computing.

4.3 Alternative possibilities

As we said, high speed of computing is one of the most important things when it comes to FEM software. Consequently, we can introduce MPI or OpenMP into our program to realize parallelization, which can considerably increase the computing speed of our program.