DTU Mekanik
DTU Compute

Project introduction

This project is proposed as a student project, where the students will work with an existing IDE program, and create a plug-in that enables the program to recognize and resolve the ambiguous usage of header files in the open source Computational Fluid Dynamics (CFD) program OpenFOAM. The header files are used for 1) Class declarations 2) self-contained pieces of code. The plug-in should distinguish between the two header types, and support the use of auto-completion, code debugging and code compilation.

Motivation

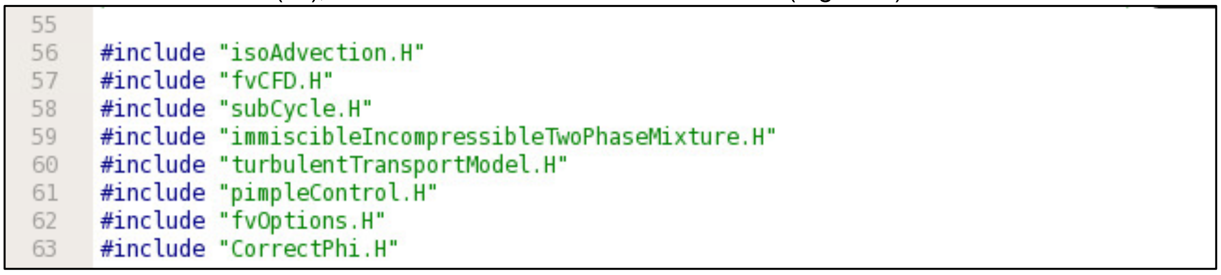
The open source program OpenFOAM is widely used within many applications of CFD in academia and in industry. The program however does not come with any interface or development environment, and it is a collection of small compiled programs that work together to form the entire program. When the user would like to modify the code or add a new capability to the code, the user needs to use a program that is called Doxygen to figure out how the program is structured, and then the user needs to change the relevant text files in a text editor.

We would like to be able to develop new code in an Integrated Developer Environment (IDE), which must be capable of understanding the coding style used in OpenFOAM. Furthermore, there is a need for a debugging application and code compilation. The latter features, i.e. debugging and compilation, already seem to be working in IDEs like Eclipse and QtCreater (Both available on G-Bar). Hence they could be the starting point of the project.

Problem description

OpenFOAM is wrottem in the object-oriented language C++. The usual coding task would be to modify an existing solver and add some additional physics to describe the physical problem at hand.

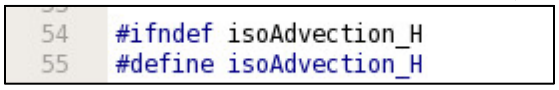
A solver is an executable program, which means that it has the main function in the (.C) file with the relevant input parameters. Before we define the main function in the C-file, we need to include a number of header files (.H), which contains the class declarations. (Figure 1)



```
#include "isoAdvection.H"
#include "fvCFD.H"
#include "subCycle.H"
#include "immiscibleIncompressibleTwoPhaseMixture.H"
#include "turbulentTransportModel.H"
#include "pimpleControl.H"
#include "fvOptions.H"
#include "CorrectPhi.H"
```

Figure 1: Included header files to define necessary class templates, object templates, etc.

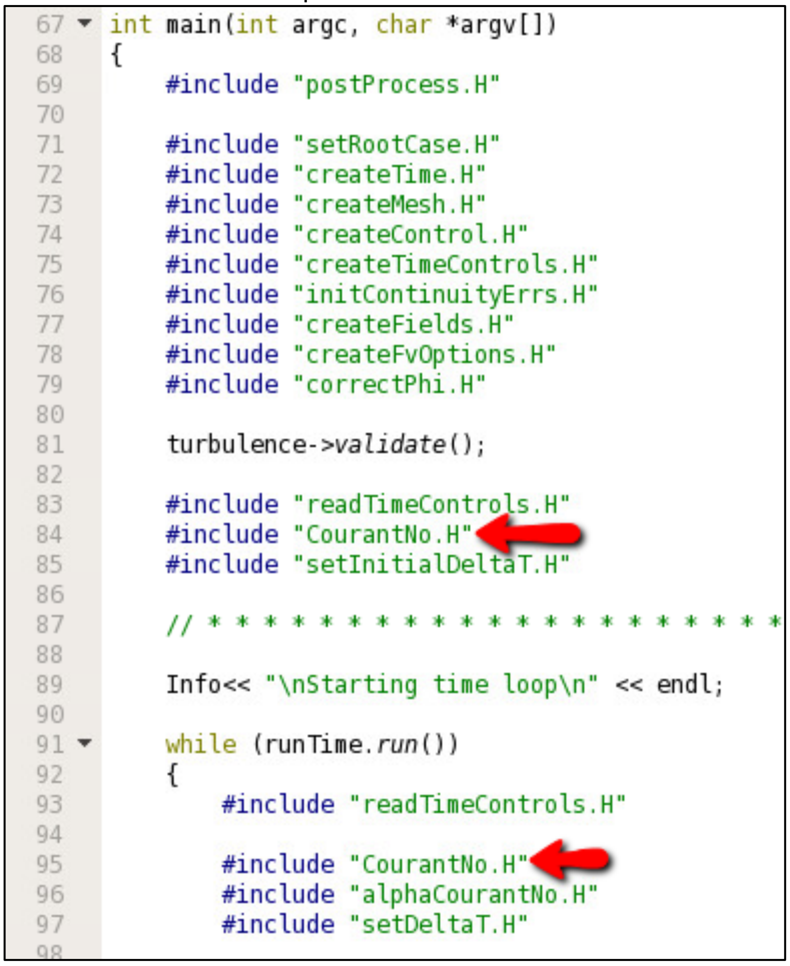
It is important to note how the above header files always are guarded against multiple includes, see Figure 2, which shows the include guard for the file “iseAdvection.H”. If any of the above header files are included more than once and did not have the include guard, the compiler would not know which definition to use from the class declarations, and the code would fail to compile.



```
#ifndef isoAdvection_H
#define isoAdvection_H
```

Figure 2: Multiple include guarding.

Let us look at the first part of the main function in Figure 3. The includes made here are of a different type, because the **header files are used to store a self-contained piece of code**, that can be included at the relevant location in the main code. This is done in order to improve readability, however IDEs such as Eclipse and QtCreater do not understand this usage of the header files, and this is the essence of the problem that we seek a solution for.

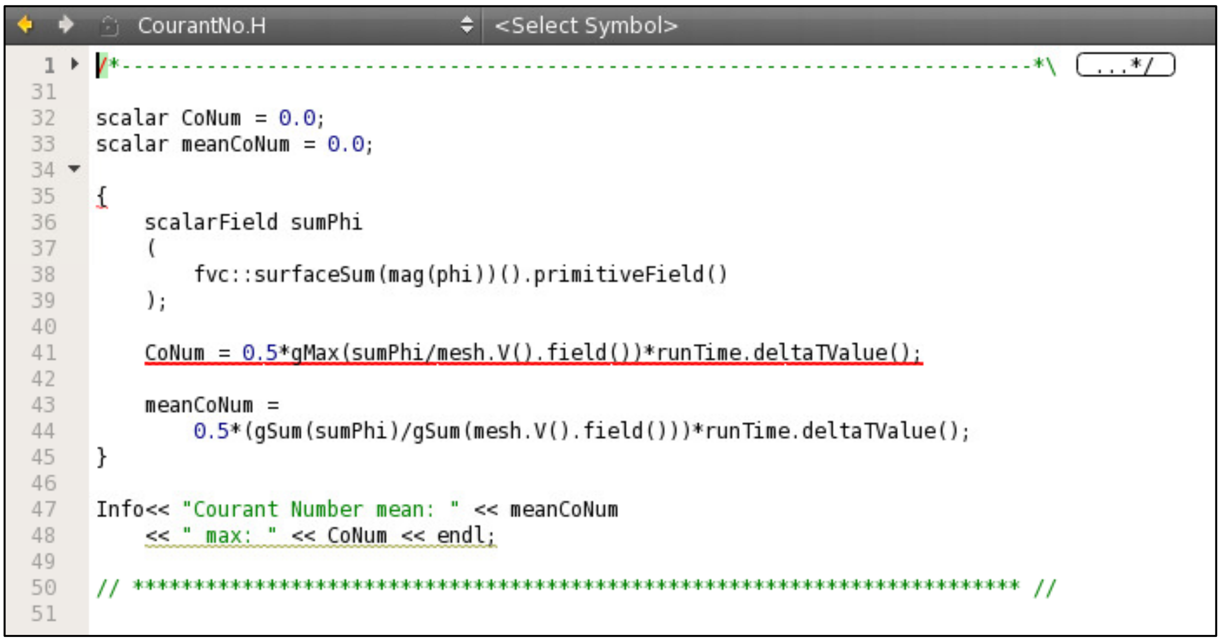


```
int main(int argc, char *argv[])
{
    #include "postProcess.H"

    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createTimeControls.H"
    #include "initContinuityErrs.H"
    #include "createFields.H"
    #include "createFvOptions.H"
    #include "correctPhi.H"

    turbulence->validate();

    #include "readTimeControls.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"

    // * * * * * * * * * * * * * * * * * * * * *

    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
        #include "readTimeControls.H"

        #include "CourantNo.H"
        #include "alphaCourantNo.H"
        #include "setDeltaT.H"
```

Figure 3: Illustration of multiple includes of ”CourantNo.H” inside the main function.

For example, the “CourantNo.H” header file contains a self-contained piece of code to calculate the courant number. This operation is carried out before the while loop and in each step of the while loop. Hence, this is clearly not a regular header file with class declarations, but instead a script file with some code that we copy in to the main function using include commands.

The content of the “CourantNo.H” file is seen in Figure 4.



```
/*---------------------------------------------------------------------------*\  ...*/

scalar CoNum = 0.0;
scalar meanCoNum = 0.0;

{
    scalarField sumPhi
    (
        fvc::surfaceSum(mag(phi))().primitiveField()
    );

    CoNum = 0.5*gMax(sumPhi/mesh.V().field())*runTime.deltaTValue();

    meanCoNum =
        0.5*(gSum(sumPhi)/gSum(mesh.V().field()))*runTime.deltaTValue();
}

Info<< "Courant Number mean: " << meanCoNum
    << " max: " << CoNum << endl;

// ************************************************************************* //
```

Figure 4: Content of "CourantNo.H" file.

We note that there is no guarding against multiple includes, as was the case for the header files with class declarations included before the main function. In addition, you may notice that QtCreator cannot recognize the syntax, because QtCreator does not know that it should use the class declarations and variables defined in the (.C) file with the main function. If I copy all the includes into the main file, this would solve the problem, but then we lose readability and we want to be able to adhere to the general coding style of OpenFOAM when developing new code.

Desired outcome

We seek a solution that works with either Eclipse or QtCreator, such that the IDE can also recognize the syntax in the header files used for self-contained pieces of code. Furthermore, we wish to be able to use auto-completion in the main file and all the header files with self-contained code. At last, we also need to be able to debug the code. The auto-completion and debugging functionalities are already in the programs; however, we need to ensure that it will also work with the new syntax recognition, which can recognize the difference between a header file used for class declarations and a header file used for self-contained code.

A possible first approach to distinguish between the two types of header files could be to recognize if the header file was guarded against multiple includes or not.

- Header file with class declarations: **Guarded.**
- Header file with self-contained piece of code: **Not guarded.**

Another possible approach is to consider as ‘Header file with self-contained piece of code’
Any file included with an #include command within structure blocks of code (e.g., within a function, a class declaration, etc) as

Supervision
Assistant Prof. Andrea Vardin: Technical supervisor
Ph.D. candidate Jesper Pedersen: End user supervision if needed.

Project type
The project could be a group of students, as well as a single student.