JKU
JOHANNES KEPLER
UNIVERSITY LINZ

Introduction to open source CFD

Session 2/4

Gijsbert Wierink

Christian – Doppler Laboratory on Particulate Flow Modelling
Johannes Kepler University | Linz | Austria

Outline

- Introduction to C++
- Modyfying a solver
- Creating our own boundary condition

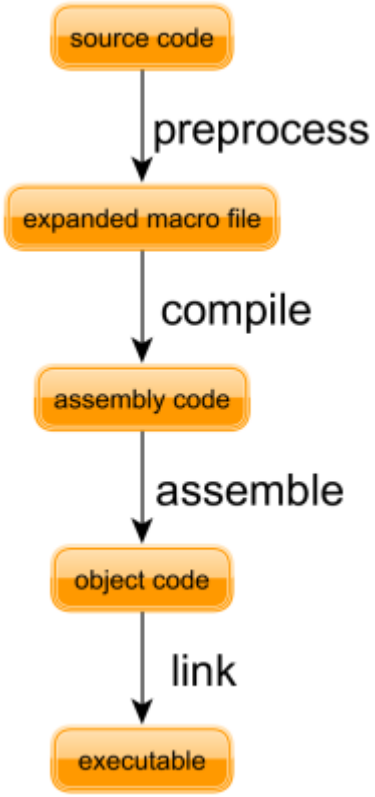
Disclaimer

This course is offered non-commercially as part of course work at the Department of Particulate Flow Modelling (PFM) at the Johannes Kepler University (JKU) in Linz, Austria. This offereing is not approved or endorsed by OpenCFD Limited, the producer of the OpenFOAM software and owner of the OPENFOAM® and OpenCFD ® trade marks.

INTRODUCTION TO C++

Introduction to C++
What the object is C++?

- C++ is a programming language that is
  - object-oriented
  - compiled
  - imperative
- OpenFOAM is a **modular** C++ code
  - Libraries such as boundary conditions, turbulence models etc., are compiled as **shared objects**
  - Executables are compiled as **(executable) objects**



Objects, functions, types and classes

- C++ works with
  - **objects** that can **hold data**
  - **methods** (functions) can **manipulate** data held by objects.

- Objects must be declared with a **type**
- A **class** is a formal **model** of how a type behaves

Introduction to C++
Objects

- C++ works with **objects** that can **hold data**. **Methods** (functions) can **manipulate** data held by objects.

- Need to create an object before you can use it:

```
type objectName = value; // pseudo code

int  a          = 4;     // copy assignment
```

Functions/methods

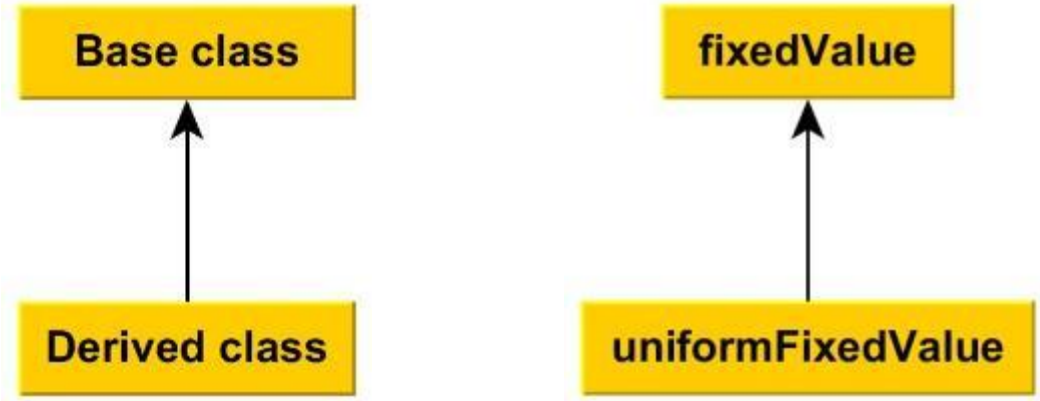
- C++ works with **objects** that can **hold data**. **Methods** (functions) can **manipulate** data held by objects.

- Functions can use an object as argument to a function, or to apply a function to:

```
operator+(a,b)

patch.size()
```

Introduction to C++
Class inheritance

- Classes can **inherit** data and/or functions from other classes.



Polymorphism

- „The ability to obtain type-specific behaviour based on the dynamic type of a reference or pointer.“[1]
- For example, a function behaves differently depending on the type of object it is applied to.

```
a  + b   // adding scalars
P1 + p2  // adding volScalarFields
U1 + U2  // adding volVectorFields
```

[1] See e.g. Chapter 15 of Lippman, S., Lajoie, J, and Moo, B., *C++ Primer*, 4th ed., Edison Wesley, 2005.

Introduction to C++
A basic example

- Create a text file named windows.C containing

```
#include <iostream>

int main()
{
    std::cout << "I love windows!" << std::endl;

    return 0;
}
```

- Compile the code by

```
g++ windows.C –o Windows
```

- Run Windows by

```
./windows
```

Introduction to C++
A basic example – OpenFOAM style

- Create a text file named `changedMyMind.C` containing

```
#include "fvCFD.H"

int main(int argc, char *argv[])
{
    Info<< "I changed my mind ..." << std::endl;

    return 0;
}
```

- We will is wmake to compile, which needs a Make directory containing `files` and `options`.

Introduction to C++
A basic example – OpenFOAM style

- Create a directory called Make in the location where the code resides

```
> mkdir Make
```

- Create a text file named `files` in the Make directory

```
> gedit Make/files &
```

- ... with the following content

```
changedMyMind.C

EXE = $(FOAM_USER_APPBIN)/changedMyMind
```

Introduction to C++
A basic example – OpenFOAM style

- ... and a file named options `options`

```
> gedit Make/options &
```

- ... With the following content

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude

EXE_LIBS = \
    -lfiniteVolume
```

A basic example – OpenFOAM style

- Finally, compile

```
> wmake
```

- ... and run

```
> changedMyMind
```

Compiling OpenFOAM code

- In short, we can compile two types of things
  - code that can be executed (solvers, utilities)

```
> wmake
```

  - code that is used by executable code (boundary conditions, turbulence models, etc.)

```
> wmake libso
```

Introduction to C++
Some resources

- Lippman, Lajoie and Moo, *C*++ Primer.
- Deitel and Deitel, C++ How to program.
- Meyers, Effective C++.
- Koenig and Moo, Accelerated C++.
- Ullman and Signer, C++ Programming: Visual QuickStart Guide.

- www.cplusplus.com
- www.learncpp.com
- https://buckysroom.org/videos.php

MODIFYING A SOLVER

Modifying a solver
Overview

Aims:

- Implement a massless species transport equation in a transient solver
- Use the new solver
- Apply a time-varying boundary condition to the species field

- We will implement a transport equation for massless species C in the incompressible RAS/LES solver `pimpleFoam`

$$\frac{\partial}{\partial t}C+\nabla\cdot\left(\vec{U}C\right)-\nabla\cdot\left(D_C\nabla C\right)=0$$

Modifying a solver
Overview

Aims:

- Implement a massless species transport equation in a transient solver
- Use the new solver
- Apply a time-varying boundary condition to the species field

Steps

- Copy the `pimpleFoam` solver to your user directory
- Rename and compile
- Implement species transport equation
- Copy standard `pimpleFoam` tutorial to run directory
- Modify the tutorial to handle the new solver

Modifying a solver
Recommended file structure

- Create a directory to store your models

```
student-2.3.x           # this is WM_PROJECT_USER_DIR
├── run                 # dir to run cases
├── src                 # dir for model libraries
└── applications
    ├── utilities
    └── solvers
```

Modifying a solver
Solver

- Go to the run directory and copy the solver

```
> mkdir -p $WM_PROJECT_USER_DIR/applications/solvers
> cd  $WM_PROJECT_USER_DIR/applications/solvers
> cp -r $FOAM_SOLVERS/incompressible/pimpleFoam pimpleSpeciesTransportFoam
> cd pimpleSpeciesTransportFoam
```

- Rename the solver

```
> mv pimpleFoam.C pimpleSpeciesTransportFoam.C
```

- Remove unneccesary directories

```
> rm -r pimpleDyMFoam SRFPimpleFoam
```

Modifying a solver
Solver

- Edit `Make/files` to look like below

```
pimpleSpeciesTransportFoam.C

EXE = $(FOAM_USER_APPBIN)/pimpleSpeciesTransportFoam
```

- … and compile

```
> wclean
> wmake
```

Modifying a solver
Solver

- Now edit `pimpleSpeciesTransportFoam.C` by adding the transport equation after the PIMPLE loop and before `runTime.write()`

```
    if (pimple.turbCorr())
    {
        turbulence->correct();
    }
}

fvScalarMatrix Ceqn
(
    fvm::ddt(C)
  + fvm::div(phi, C)
  - fvm::laplacian(DC, C)
);

CEqn.solve();

runTime.write();
```

Modifying a solver
Solver

- Create the necessary objects in `createFields.H`. First, create species field C

```
Info<< "Reading field C\n" << endl;

volScalarField C
(
    Ioobject
    (
        "C",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
```

Modifying a solver
Solver

- Create the necessary objects in `createFields.H`. Then, read the diffusion coefficient from a dictionary

```
Info<< "Reading transportProperties\n" << endl;
IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);

dimensionedScalar DC
(
    transportProperties.lookup("DC")
);
```

Modifying a solver
Solver

- Adjust the header to detail your changes

- Compile the solver

```
> wmake
```

Modifying a solver
Test case

- Let's set up our beloved `pitzDaily` case for our new solver
- Go to the run directory and copy the `pimpleFoam pitzDaily` case in there

```
> run
> cp -r \
  $FOAM_TUTORIALS/incompressible/pimpleFoam/pitzDaily \
  pitzDailySpeciesTransport
> cd pitzDailySpeciesTransport
```

Modifying a solver
Test case

- Now create a new field `C` in the `0` directory

```
> cp 0/p 0/C
```

- … and edit the `0/C` file so that the content looks like

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      C;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;
```

Modifying a solver
Test case

- Add our new variable to `system/fvSolution`

```
"(U|k|epsilon|C)"
{
    solver          smoothSolver;
    smoother        symGaussSeidel;
    tolerance       1e-05;
    relTol          0.1;
}

"(U|k|epsilon|C)Final"
{
    $U;
    relTol          0;
}
```

Modifying a solver
Test case

- … and to `system/fvSchemes`

```
divSchemes
{
    default         none;
    ...
    div((nuEff*dev(T(grad(U))))) Gauss linear;
    div(phi,C)      bounded Gauss upwind;
}

laplacianSchemes
{
    default         none;
    ...
    laplacian(DC,C) Gauss linear corrected;
    laplacian(DnuTildaEff,nuTilda) Gauss linear corrected;
}
```

Modifying a solver
Test case

- Finally, change the kinematic viscosity and add the diffusion coefficient in `constant/transportProperties`

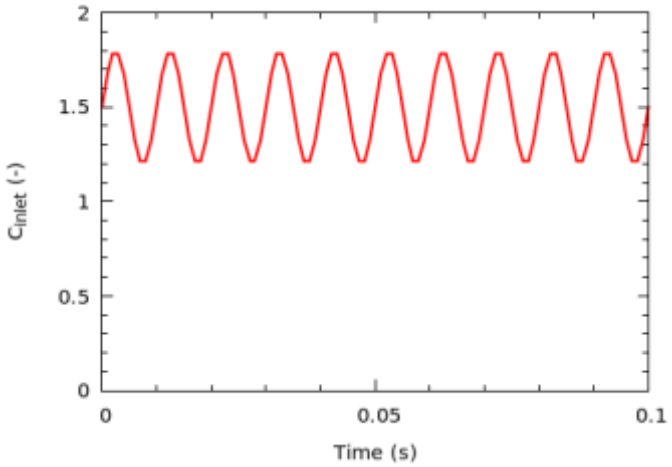
```
nu              nu [ 0 2 -1 0 0 0 0 ] 1e-6;

DC              DC [ 0 2 -1 0 0 0 0 ] 1e-7;
```

Modifying a solver
Test case

- Just because it is fun, use an oscillating inlet BC for the species field `C`. Edit `0/C`:

$$\phi_p = (1 + a \sin(2\pi f))\phi_{ref} + \phi_0$$

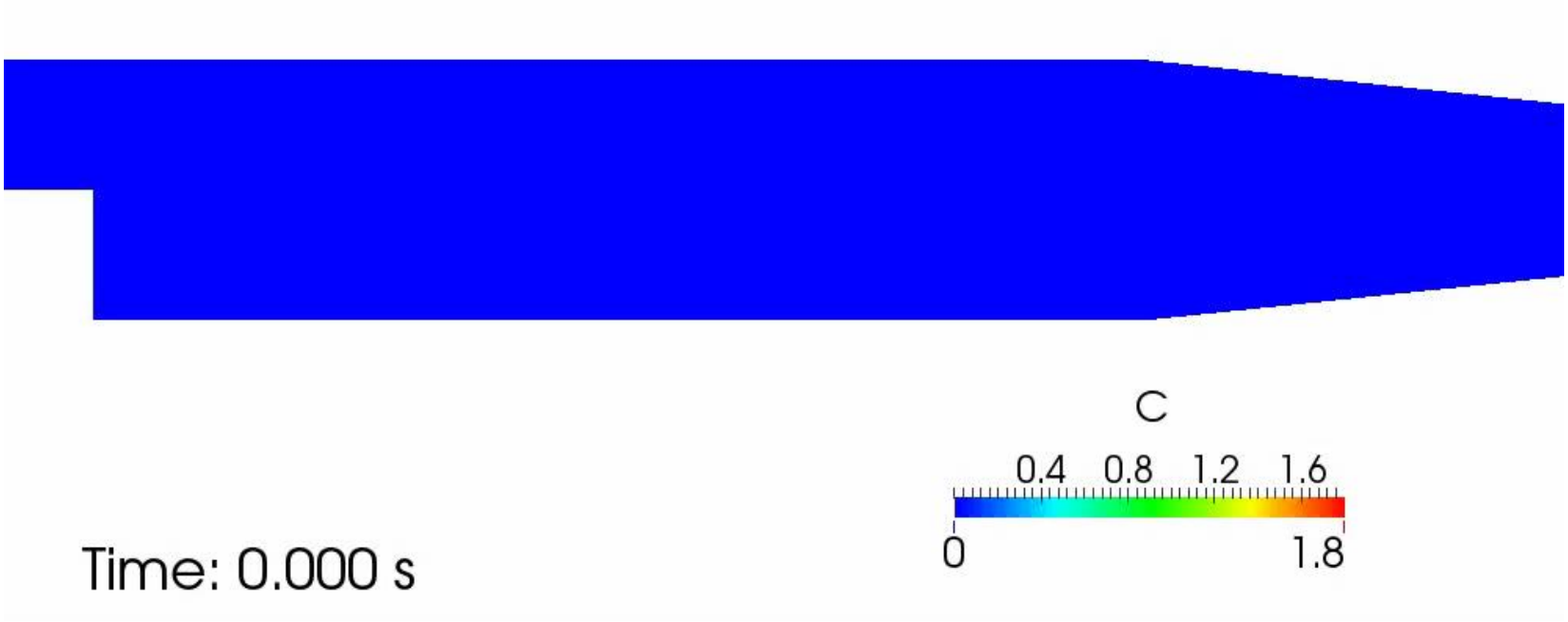
```
boundaryField
{
    inlet
    {
        type            oscillatingFixedValue;
        refValue        uniform 1.0;
        offset          0.5;
        amplitude       constant 0.3;
        frequency       constant 100;
    }
```



Modifying a solver
Test case

```
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    upperWall
    {
        type            zeroGradient;
    }
    lowerWall
    {
        type            zeroGradient;
    }
    frontAndBack
    {
        type            empty;
    }
}
```

Modifying a solver
Test case

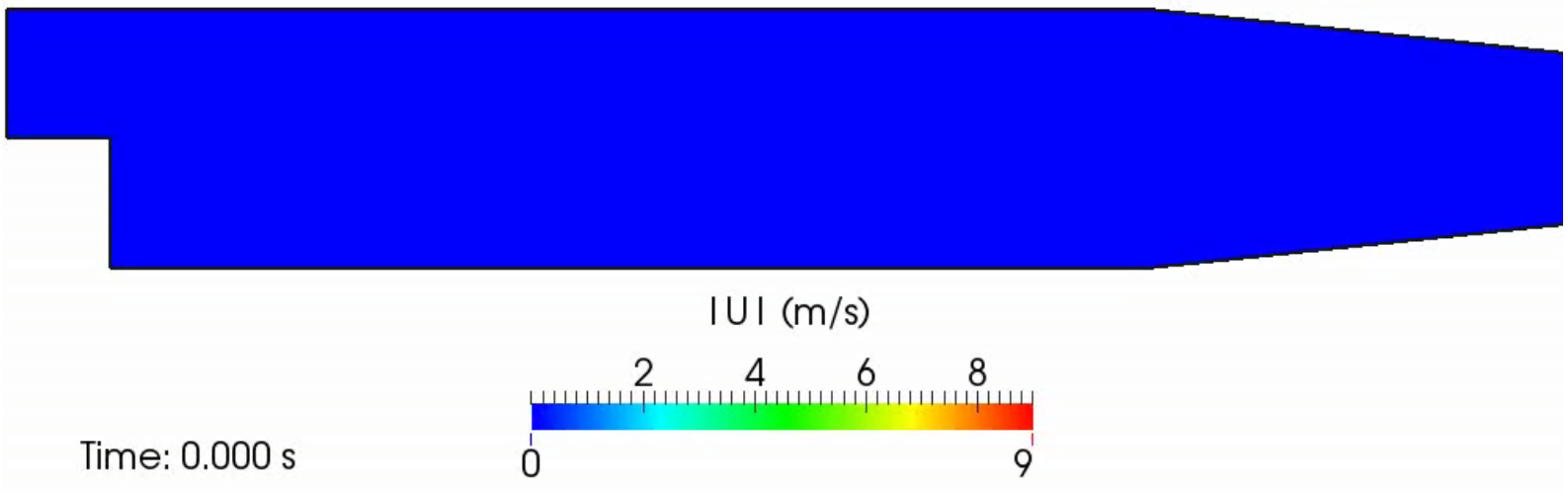


CREATING A BOUNDARY CONDITION

Creating a boundary condition (BC)
Overview

Aims:

- Create a library
- Create a new boundary condition
- Use the new BC on a case



Creating a boundary condition (BC)
Overview

- Let's have a look what BCs are available

```
> find $FOAM_SRC -name "*vPatchFields"
```

- Look more specifically at the common FVM BCs

```
> src

> cd finiteVolume/fields/fvPatchFields

> ls -1
basic
constraint
derived
doc
fvPatchField
```

Creating a boundary condition (BC)
Find an example to start from

- Create a boundary condition that randomises inlet flow rate

- First, let's see whether we can find a good starting point:

```
> src
> cd finiteVolume/fields/fvPatchFields/derived
> ls
```

Creating a boundary condition (BC)

- Create a directory to store your models

```
student-2.3.x              # this is WM_PROJECT_USER_DIR
├── run                    # dir to run cases
├── src                    # dir for model libraries
└── applications
    ├── utilities
    └── solvers
```

- … and copy the flowRateInletVelocity BC

```
> cd $WM_PROJECT_USER_DIR
> mkdir -p src/fvPatchFields
> cd src/fvPatchFields
> BCS="$FOAM_SRC/finiteVolume/fields/fvPatchFields/derived"
> cp -r $BCS/flowRateInletVelocity randomisedFlowRateInletVelocity
> cp randomisedFlowRateInletVelocity
```

Creating a boundary condition (BC)

- First, rename the file names (on debian-ish system!)[1]

```
> rename 's/flowRateInletVelocity/randomisedFlowRateInletVelocity/g' *.*
```

- Then, change the instances inside the files

```
> sed -i 's/flowRateInletVelocity/randomisedFlowRateInletVelocity/g' *.*
```

- … using sed. Some explanation on sed syntax:

```
sed -i 's/string1/string2/g' file
sed: stream editor
-i:  "in place" (as in "insert here")
s:   substituteg: append pattern space ("global")
```

[1] On an rpm-ish system the syntax for rename is:

```
> rename string1 string2
```

Creating a boundary condition (BC)

- The flowRateInletVelocity BC does not have a Make directory, so we need to create it

```
> mkdir Make
```

- … and create `Make/files` as

```
randomizedFlowRateInletVelocityFvPatchVectorField.C

LIB = $(FOAM_USER_LIBBIN)/libfancyStuff
```

Creating a boundary condition (BC)

- … and create `Make/options` as

```
EXE_INC = \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/postProcessing/foamCalcFunctions/lnInclude \
    -I$(LIB_SRC)/OpenFOAM/lnInclude

LIB_LIBS = \
    -lfiniteVolume \
    -lfoamCalcFunctions \
    -lOpenFOAM
```

- Finally, compile before making code changes

```
> wmake libso
```

Creating a boundary condition (BC)

- Add necessary headers to `randomizedFlowRateInletVelocityFvPatchVectorField.H`

```
#include "DataEntry.H"

#include "calcType.H"
#include "Random.H"

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

Creating a boundary condition (BC)

- Add private data to the class declaration in `randomizedFlowRateInletVelocityFvPatchVectorField.H`

```
        //- Rho initialisation value (for start; if value not supplied)
        scalar rhoInlet_;

        //- Random number generator
        Random ranGen_;

        //- Fluctuating scale
        scalar fluctuationScale_;

    public:
```

Creating a boundary condition (BC)

- Initialize randGen_ and fluctuationScale_ in the constructors in `randomizedFlowRateInletVelocityFvPatchVectorField.C`

- First and second constructors

```
    rhoInlet_(0.0),
    ranGen_(label(0)),
    fluctuationScale_(0.0)
{}
```

- Third constructor

```
    rhoInlet_(dict.lookupOrDefault<scalar>("rhoInlet", -VGREAT)),
    ranGen_(label(0)),
    fluctuationScale_(readScalar(dict.lookup("fluctuationScale")))
{
    if (dict.found("volumetricFlowRate"))
```

Creating a boundary condition (BC)

- Initialize randGen_ and fluctuationScale_ in the constructors in `randomizedFlowRateInletVelocityFvPatchVectorField.C`

- Fourth and fifth constructors

```
    rhoInlet_(ptf.rhoInlet_),
    ranGen_(ptf.ranGen_),
    fluctuationScale_(ptf.fluctuationScale_)
{}
```

Creating a boundary condition (BC)

- Now modify the `updateCoeffs` function in `randomizedFlowRateInletVelocityFvPatchVectorField.C`

```
// a simpler way of doing this would be nice
const scalar avgU = -flowRate_->value(t)/gSum(patch().magSf());

fvPatchVectorField& patchField = *this;

scalarField randomField(this->size());

forAll(patchField, facei)
{
    ranGen_.randomise(randomField[facei]);
}
```

Creating a boundary condition (BC)

- Now modify the `updateCoeffs` function in `randomizedFlowRateInletVelocityFvPatchVectorField.C`

```
scalarField rndScale =
    pTraits<scalar>::one + fluctuationScale_*randomField;

tmp<vectorField> n = patch().nf();

if (volumetric_ || rhoName_ == "none")
{
    // volumetric flow-rate or density not given
    operator==(n*avgU*rndScale);
}
```

Creating a boundary condition (BC)

- Last, but not least, update the `write` function in `randomizedFlowRateInletVelocityFvPatchVectorField.C`

```
void Foam::randomizedFlowRateInletVelocityFvPatchVectorField::write
(
    Ostream& os
) const
{
    fvPatchField<vector>::write(os);
    flowRate_->writeData(os);
    os.writeKeyword("fluctuationScale")
        << fluctuationScale_ << token::END_STATEMENT << nl;
    if (!volumetric_)
    {
        writeEntryIfDifferent<word>(os, "rho", "rho", rhoName_);
        writeEntryIfDifferent<scalar>(os, "rhoInlet", -VGREAT, rhoInlet_);
    }
    writeEntry("value", os);
}
```

Creating a boundary condition (BC)

- Finally, compile the library

```
> wmake libso
```

- Change to the run directory and copy a tutorial to test the BC

```
> run
> cp -r \
  $FOAM_TUTORIALS/incompressible/pimpleFoam/pitzDaily \
  pitzDailyRandomisedInlet
> cd pitzDailyRandomisedInlet
```

Creating a boundary condition (BC)

- Apply the new BC to the inlet patch in `0/U`

```
inlet
{
    type            randomizedFlowRateInletVelocity;
    volumetricFlowRate  0.002;
    fluctuationScale    0.1;
    value           uniform (0 0 0); // placeholder
}
```

- … and add your model library to the run-time selection table by adding the following to `system/controlDict`, e.g. at top or bottom of file

```
libs
(
    "libOpenFOAM.so"
    "libfancyStuff.so"
);
```

Creating a boundary condition (BC)

- Run the solver

```
> pimpleFoam
```