

Basics of C++ and object orientation in OpenFOAM

- To begin with: The aim of this part of the course is not to teach all of C++, but to give a short introduction that is useful when trying to understand the contents of OpenFOAM.
- After this introduction you should be able to *recognize* and make *minor modifications* to most C++ features in OpenFOAM.
- ~~We will follow the book *C++ direkt* by Jan Skansholm (ISBN 91-44-01463-5)~~

C++ basics, Types

- **Types** define what values a variable may obtain, and what operations may be made on the variable.
- Pre-defined C++ types are:

signed char	unsigned int
short int	unsigned long int
int	float
unsigned char	double
unsigned short int	long double

- User defined types can be defined in **classes**. OpenFOAM provides numerous types that are useful for solving partial differential equations.
- **OpenFOAM classes are used by including the class declarations in the header of the code, and linking to the corresponding compiled OpenFOAM library at compilation.**
- The path to included files that are in another path than the current directory must be **specified by -I**
- **The path to libraries that are linked to is specified with -L**

C++ basics, object orientation

- Object orientation focuses on the *objects* instead of the functions.
- The *types* that we have just had a look at are in fact *classes*, and the variables we assign to a *type* are objects of that class.
- An *object* belongs to a *class* of objects with the same attributes. The class defines the construction of the object, destruction of the object, attributes of the object and the functions that can manipulate the object. I.e. it is the `int` class that defines how the operator `+` should work for objects of that class, and how to convert between classes if needed.
- The objects may be related in different ways, and the classes may inherit attributes from other classes.
- A benefit of object orientation is that the *classes can be re-used*, and that each class can be designed and bug-fixed for a specific task.
- The classes can be seen as new types that are designed for specific tasks.
- In OpenFOAM, the *classes are designed to define, discretize and solve PDE's*.

C++ basics, using a class

- An object of a class name is defined in the main code as:
`name nameObject; (c.f. int i)`
- `nameObject` will then have all the attributes defined in the class `name`.
- Any number of objects may belong to a class, and the attributes of each object will be separated.
- There may be pointers and references to any object.
- The member functions operate on the object according to its implementation.
If there is a member function `write` that writes out the contents of an object of the class `name`, it is called in the main code as:

```
nameObject.write();
```

- When using the memberfunctions through a pointer, the syntax is slightly different (here `p1` is a pointer to the object `nameObject`, and `p2` is a pointer to a nameless new `name`):

```
p1 = &nameObject;  
p2 = new name;  
p1->write();  
p2->write();
```

C++ basics, member functions

- The member functions may be defined either in the *definition of the class*, or in the *declaration of the class*. We will see this when we look inside OpenFOAM. The syntax is basically:

```
inline void name::write()  
{  
    Contents of the member function.  
}
```

where `name::` tells us that the member function `write` belongs to the class `name`, `void` tells us that the function *does not return anything*, and `inline` tells us that the function will be *inlined* into the code where it is called instead of jumping to the memory location of the function at each call (good for small functions). Member functions defined directly in the class definition will automatically be inlined if possible.

- The member functions have direct access to all the data members and member functions *of the class*.

C++ basics, organization of classes

- A good programming standard is to make the class files in pairs, one with the class definition, and one with the function declarations.
- Classes that are closely related to each other can share files, but keep the class definitions and function declarations separate. This is done throughout OpenFOAM.
- The class ~~definitions~~ must be included in the object file that will use the class, and in the function ~~n declarations~~ file. The object file from the compilation of the declaration file is statically or dynamically linked to the executable by the compiler.
- Inline functions must be implemented in the class ~~definition~~ file, since they must be inlined without looking at the function ~~declaration~~ file. In OpenFOAM there are usually files named as `VectorI.H` containing inline functions, and those files are included in the corresponding `Vector.H` file.
- Let's have a look at some examples in the OpenFOAM `Vector` class:
`$FOAM_SRC/OpenFOAM/primitives/Vector`

C++ basics, Constructors

- A constructor is a special initialization function that is called each time a new object of that class is defined. Without a specific constructor all attributes will be undefined. A null constructor must always be defined.
- A constructor can be used to initialize the attributes of the object. A constructor is recognized by it having the same name as the class - here `Vector`. (`Cmpt` is a template generic parameter (*component type*), i.e. the `Vector` class works for all component types). `Vector.H`:

```
// Constructors
// - Construct null
inline Vector();
// - Construct given VectorSpace
inline Vector(const VectorSpace<Vector<Cmpt>, Cmpt, 3>&);
// - Construct given three components
inline Vector(const Cmpt& vx, const Cmpt& vy, const Cmpt& vz);
// - Construct from Istream
inline Vector(Istream&);
```

- The `Vector` will be initialized differently depending on which of these constructors is chosen.

C++ basics, typedef

- OpenFOAM is full of templates.
- To make the code easier to read OpenFOAM re-defines the templated class names, for instance:

```
typedef List<vector> vectorList;
```

so that an object of the class `template List` of type `vector` is called `vectorList`.

