Школа-семинар

«Расширенные возможности пакета OpenFOAM»

C++ ЗА 8 ШАГОВ

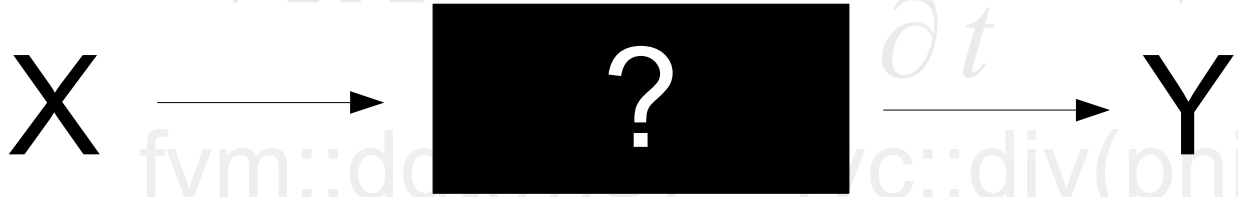
М.В. Крапошин (НИЦ Курчатовский Институт)
О.И. Самоваров (Институт Системного Программирования РАН)
С.В. Стрижак (ГОУ ВПО МГТУ им. Баумана)

СОДЕРЖАНИЕ РАЗДЕЛА

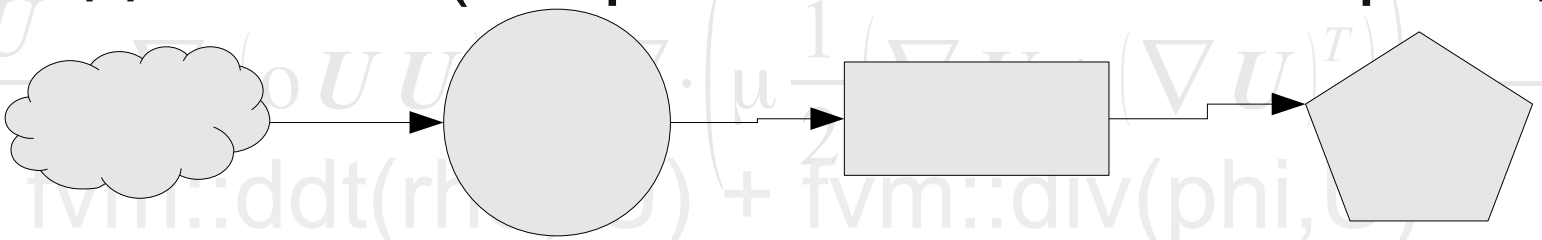
- Объектно-ориентированное программирование
- Базовые типы данных
- Специальные типы данных
- Классы (пользовательские типы данных)
- Область видимости
- Шаблоны
- Преобразование типов
- Язык препроцессора
- Обработка исключительных ситуаций

ТРИ КИТА ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

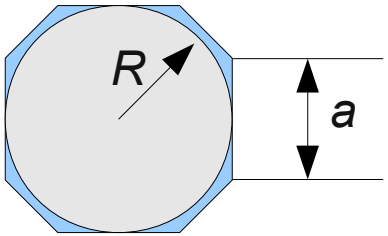
- Инкапсуляция (сокрытие данных)



- Наследование (сохранение базовых принципов)



- Полиморфизм (плюрализм методов решения)



$$S = \pi R^2 = \pi \frac{D^2}{4} = \frac{1}{4\pi} l^2 \approx n \times a$$

ОТЛИЧИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО (C++) ОТ ПРОЦЕДУРНОГО СТИЛЯ ПРОГРАММИРОВАНИЯ (F77)

Процедурное Программирование. Базовые единицы — переменная и функция. Стратегия программы определяется до написания исходного в виде блок-схем или псевдокода. Пример: Fortran, C, Pascal

```
subroutine sum (a,b,c)
  real*4 a, b, c
  c = a + b
end subroutine
```

Объектно-ориентированное программирование. Базовая единица — класс, содержащий в себе и данные (атрибуты) класса и методы

```
class A{
  float val;
  const A& sum(const A& a, const A& b){
    val = a.val + b.val;
    return *this;
  }
};
```

C++ за 8 шагов (I)

Шаг 1. Базовые типы переменных, присваивание, ветвление, циклы

1) Переменные всегда объявляются явно. Поддерживаются следующие базовые типы:
short, int, long, float, double, char, bool, void

```
double a=0.0,b; int l=1,k; bool c; const float pi=3.14;
```

2) Присваивание, присваивание с инкрементом и декрементом: **=, +=, -=, *=, /=, ++, - -**

```
s=0; s=s+1; s+=1; s++; a=b=s++; a=b=++s; a+=b;
```

3) Ветвление: конструкции **if-else if, switch**

```
if (a>=0 && a<=1)
{ // если a в диапазоне[0,1]
}
else if (a<0){} //если a < 0
else{} //все остальные случаи
```

```
switch (j)
{
 case 0: {break;}// если j = 0
 case 1: {break;}// если j = 1
 default: {break;}// другие случаи
}
```

4) Циклы: while, for

```
while (i < 10)
{i++;}
do
{j++;} while (j<10);
```

```
for (int j=0; j<10; j++)
{
 cout << j << endl;
}
```

```
while(1){}
for(;;){}
```

БАЗОВЫЕ ТИПЫ ДАННЫХ OPENFOAM

№№	Тип OpenFOAM	Тип C++	Описание
1	`label`	`int, long`	Целые положительные и отрицательные числа, label.H
2	`scalar`	`double (float)`	Числа с плавающей точкой (двойной или одинарной точности), scalar.H
3	`word`	`char *, string`	Символьная строка (с ограничением по некоторым символам), word.H
4	`Switch`	`bool`	Логическое значение, Switch.H

Некоторые другие базовые типы: List, vector, tensor, dimensioned

БАЗОВЫЕ ТИПЫ ДАННЫХ OPENFOAM

Для операций с базовыми типами в OpenFOAM определены практически все базовые операторы — присваивания, сравнения, арифметические

```
#include <scalar.H>
#include <label.H>
#include <Switch.H>
#include <word.H>
#include <cmath>
#include <iostream>

using namespace Foam;
```

```
int main (int argc, char * argv[])
{

    label i = 1;
    int   j = 2;
    label k = i;

    scalar pi1 = 3.14;
    double d   = M_PI - pi1;

    Switch eq = (d <= 0.0);

    word goodStr = "abc";
    word badStr  = "abc abc";

    return 0;
}
```

ВЕТВЛЕНИЕ, ЦИКЛЫ, СТИЛЬ КОДА

for — цикл с заданными числом итераций, while — цикл с прерыванием по условию

```
//for example
scalar sum = 0.0;
label N = 10;
for (label j=0; j<N; j++)
{
    sum += j;
}

//do-while example
label k = N;
do
{
    k--;
    sum -= k;
}
while ( k > 0);
```

if-else if-else — выполнение блоков кода по условию

```
//if example
if (sum == 0)
{
    k = 0;
}
else if (sum > 0)
{
    k = 1;
}
else
{
    k = -1;
}
```

ВЕТВЛЕНИЕ, ЦИКЛЫ, СТИЛЬ КОДА

```
switch (k)
{
    case 0:
    {
        sum = 0.0;
        break;
    }
    case 1:
    {
        sum = 1.0;
        break;
    }
    case -1:
    {
        sum = -1.0;
        break;
    }
    default:
    {
        sum = VSMALL;
    }
}
```

switch-case — выполнение блоков исходного кода по «переключателю»

forAll — макрос OpenFOAM для автоматического итерирования по элементам списков OpenFOAM

```
//forAll example
List<label> l(N);
forAll (l, i)
{
    l[i] = 2*i;
}
```

ВЕТВЛЕНИЕ, ЦИКЛЫ, СТИЛЬ КОДА

- В OpenFOAM-1.6-ext в требования к оформлению исходного кода содержатся в doc/GUIDELINES
- Или во вспомогательных файлах курса — Files/day1_CPlusPlus/CodingStyle
- Перед содержимым между фигурными скобками — отступ (4 пробела или один tab)
- Имена классов и переменных — с маленькой буквы, в сложных именах — каждое новое слово с большой
- Длина кода в функциях — не более двух страниц (160 строк)
- Длина строки должна быть ограничена 25 символами

C++ за 8 шагов (II)

Шаг 2. Специальные типы переменных. Указатели, массивы , ссылки, функции

1) Указатель * — переменная, хранящая адрес некоторой области памяти заданной длины, которая определяется типом указателя либо вручную (при типе указателя void или в случае массива). К указателям применимы операции =, +, -, ++, - -, +=, -= и прочие

```
double a=0.0; double *pa = &a; *pa+=10;
```

2) Массив [] — статический эквивалент указателя — хранит указатель на первый элемент и длину массива (число элементов)

```
double a[2]; a[0] = 1.0; *(a+1) = a[0]*2.0;
```

3) Ссылка **&** - символическая ссылка на переменную, с которой можно работать как с самой переменной. Безопасный аналог указателя.

```
double a=0.0; double &la = a; la+=10;
```

4) Функция **()** - поименованный поток инструкций, в который передаются данные и из которого возвращаются данные.

```
double power (double& val, int f){
  double v = 1.0;
  for (int i=0; i<f; i++)
    v*=val;
  val = v; return v;}
```

СПЕЦИАЛЬНЫЕ ТИПЫ OPENFOAM

Классы контейнеры (реализованы с помощью механизма шаблонов C++). Предназначены для хранения объектов в динамической памяти:

- Программные указатели — autoPtr
- Программные указатели с подсчетом ссылок — tmp (объект должен наследовать от refCount)
- Хранение данных — Xfer
- Списки (массивы) — UList, List, DynamicList

СПЕЦИАЛЬНЫЕ ТИПЫ OPENFOAM

```
//pointers
label i = 1;
label& li = i;
label *pi = &i;
*pi++;
autoPtr<label> ap;
ap.reset(pi);
label *pi2 = ap.ptr();

//arrays
scalar da[20];
scalar * db;
db = new scalar [20];
for (label q=0; q<20; q++)
{
    da[q] = q;
    *(db+q) = mult(da[q], q);
}
delete db;

return 0;
```

autoPtr<> — хранение указателя с функциями автоматической очистки памяти

В C++ массив — указатель на область памяти, с длиной равной числу объектов

```
scalar mult (scalar a, scalar b)
{
    return a*b;
}
```

C++ за 8 шагов (III)

Шаг 3. Класс — новый тип данных, содержащий в себе как данные, так и методы их обработки (не путать со structure из языка C)

Ключевые составляющие: член, метод, область видимости (**private, protected, public**), конструктор, деструктор, оператор, виртуальные методы и друзья, статические единицы

```
class Shape{
  private:
    int id_;
  protected:
    double surface_;
    double center_;
  public:
    Shape();
    Shape(const Shape&);
    ~Shape();
    virtual double surface();
    virtual double center();
    virtual const Shape& operator = (const Shape& s);
    static double Pi();
    friend double anyFunc();
};
```

Члены класса — переменные, методы — функции и операторы

private — члены класса видны только внутри методов класса
protected — члены класса видны только внутри методов этого класса и всех от него наследующих
public — доступны из любого места кода

КЛАССЫ C++ В OPENFOAM (1)

- Механизм наследования и переопределения

```
class Shape : public refCount
{
public:
    Shape();
    Shape(const Shape&);
    virtual ~Shape();
    virtual scalar area()
    const;
    virtual scalar centerX()
    const = 0;
    virtual scalar centerY()
    const = 0;
};
```

```
class Circle : public Shape
{
private:
    scalar x_;
    scalar y_;
    scalar r_;
public:
    Circle();
    Circle(scalar rx, scalar ry, scalar r);
    Circle(const Circle&);
    virtual ~Circle();
    virtual scalar area() const;
    virtual scalar centerX() const;
    virtual scalar centerY() const;
    virtual void   centerX(scalar x);
    virtual void   centerY(scalar y);
};
```

Наследование — повторное использование свойств базового класса

КЛАССЫ C++ В OPENFOAM (2)

Выделение памяти — оператор **new** — конструктор класса

Class1::Class1(...){ ….}

Class1 * obj = new Class1 (...)

```
Foam::Shape::Shape()
: refCount()
{}
Foam::Shape::Shape (const Shape& s)
: refCount()
{}
Foam::Shape::~Shape()
{}
```

```
Foam::Circle::Circle()
: Shape(), x_(0.0), y_(0.0), r_(0.0)
{}

Foam::Circle::Circle(scalar x, scalar y,
scalar r)
: Shape(), x_(x), y_(y), r_(r)
{}

Foam::Circle::Circle(const Circle& c)
: Shape (c), x_(c.x_), y_(c.y_), r_(c.r_)
{}
```

Освобождение памяти — оператор **delete** — деструктор класса

Class1::~Class1(){ ….}

delete obj;

ПРИМЕР КЛАССА: КОМПЛЕКСНОЕ ЧИСЛО В OPENFOAM

- Класс complex комплексных чисел

```
class complex
{
    scalar re, im;
public:
    typedef complex cmptType;
    static const char* const typeName;
    static const complex zero;
    static const complex one;
    inline complex();
    inline complex(const scalar Re, const scalar Im);
    complex(Istream&);
    inline scalar Re() const;
    inline scalar Im() const;
    inline scalar& Re();
    inline scalar& Im();
    inline complex conjugate() const;
    inline const complex& operator=(const complex&);
    inline void operator+=(const complex&);
    inline void operator-=(const complex&);
    inline void operator*=(const complex&);
    inline void operator/=(const complex&);
```

Содержит переопределение основных операторов C++ для упрощения использования в выражениях исходного кода.

В данном случае — полная совместимость с базовыми типами данных (scalar, label)

C++ за 8 шагов (IV)

Шаг 4. Область видимости — время жизни объекта, пространства имен, доступ к членам и методам класса

В зависимости от целей локализации данных могут быть следующие способы классификации области видимости:

- По времени жизни

```
int a, b;
{
  int c, d; {int e,f;} for (int j=0; j<10; j++)
{}
}; int c, d;
```

- По пространству имен

```
namespace ns1{
 class A{};
}
```

- По уровню доступа к структурным единицам класса:
  private — доступ только из методов данного класса, либо класса, либо наследующего со спецификатором private
  protected — доступ из методов данного класса и всех наследующих
  public — доступ из тела программы и любых методов класса
  friend — позволяет получить доступ к private и protected без наследования

ОБЛАСТИ ВИДИМОСТИ В OPENFOAM

- Примеры пространств имен: Foam (все классы OpenFoam), fvc (функции Foam::fvc::***), fvm (функции Foam::fvm::***), compressible (классы Foam::compressible для моделей турбулентности)
- movingMeshContinuityErrs.H

```
{
    volScalarField contErr = fvc::div(phi + fvc::meshPhi(U));
    scalar sumLocalContErr = runTime.deltaTValue()*
        mag(contErr)().weightedAverage(mesh.V()).value();
    scalar globalContErr = runTime.deltaTValue()*
        contErr.weightedAverage(mesh.V()).value();
    cumulativeContErr += globalContErr;
    Info<< "time step continuity errors : sum local = " <<
       sumLocalContErr      << ", global = " << globalContErr
        << ", cumulative = " << cumulativeContErr
        << endl;
}
```

C++ за 8 шагов (V)

Шаг 5. Шаблоны — обобщение классов на препроцессорном уровне (до трансляции в машинный код)

Шаблоны используются при обобщении операций с различными типами данных, в которых класс объекта может варьироваться на стадии написания программы

```
template <class T> class vector
{
  private:
    T * data_;
    int size_;
  public:
    vector (int size);
    const T& operator [] (int idx);
    T& operator (int idx);
    const vector<T>& operator = (const vector<T>& v);
}
```

ПРИМЕР ШАБЛОНА - List

В процессе трансляции параметры шаблона подставляются и фактически генерируется новый тип данных

```
template<class T> class List :   public UList<T>
{
public:
        inline List();
        explicit List(const label);
        List(const label, const T&);
        List(const List<T>&);
        List(Istream&);
        inline autoPtr<List<T> > clone() const;
        ~List();
         inline void resize(const label);
         inline void resize(const label, const T&);
         void clear();
         inline void append(const T&);
         inline void append(const UList<T>&);

        void operator=(const UList<T>&);
        void operator=(const List<T>&);
```

C++ за 8 шагов (VI)

Шаг 6. Преобразование типов RTTI (Real Time Type Identification)

Язык C++ обладает средствами контроля типа переменной:

• **const_cast<T> (…)** - отказ от константности объекта

• **static_cast<T> (…)** - Преобразует выражение к указанному типу, без каких-либо проверок во время выполнения программы

• **dynamic_cast<T> (…)** - Преобразует выражение (ссылку на объект класса или указатель) к указанному типу, с проверкой во время выполнения программы, является ли выражение ссылкой или указателем на объект класса, эквивалентного или производного от того, что указан

• **reinterpret_cast<T> (…)** - Позволяет преобразовать выражение, являющееся указателем любого типа, к указанному типу

ПРЕОБРАЗОВАНИЕ ТИПОВ, objectRegistry

- Функция lookupObject (использует reinterpert_cast)

```
const volScalarField& constp = mesh.thisDb().lookupObject
                               <volScalarField>("p");
```

- const_cast (отказ от константности)

```
volScalarField& p = const_cast<volScalarField&>(constp);
```

- Функция isA (определение типа)

```
if (isA<volScalarField>(p))
{
////////////
}
```

C++ за 8 шагов (VII)

Шаг 7. Язык препроцессора (условная компиляция)

1) Условие #ifdef (Если переменная объявлена)
2) Условие #ifndef (Если переменная не объявлена)
3) Условие #else (Прочие значения)
4) Завершение ветвления #endif
5) Объявление #define или -D
6) Макрообъявление #define MACROS (PARAM1,PARAM2)
8) Предупреждение #warning
9) Ошибка #error

```
#ifndef VAR1
#define VAR1
#endif
```

МАКРООБЪЯВЛЕНИЯ В КОДЕ OPENFOAM

1) Ограничение рекурентности при подключении заголовочных файлов

2) Определение повторяемых частей кода с параметрами (аналогично шаблону, но используется для алгоритмов, шаблоны — для классов) — **forAll** (итерация по элементам списка), **TypeName** (задание символьного имени класса), **defineTypeNameAndDebug**, **defineTemplateTypeNameAndDebug** (объявление имени класса в символьном виде)

МАКРООБЪЯВЛЕНИЯ В КОДЕ OPENFOAM

Ограничение рекурентности

```
#ifndef turbulenceModel_H
#define turbulenceModel_H

#include "primitiveFieldsFwd.H"
#include "volFieldsFwd.H"
#include "surfaceFieldsFwd.H"
#include "fvMatricesFwd.H"
.............
class turbulenceModel
:
    public regIOobject
{
...............

#endif
```

Повторяющиеся части кода

```
// Declare a ClassName() with extra
// virtual type info

#define TypeName(TypeNameString)\
    ClassName(TypeNameString); \
    virtual const word& type() const
{ return typeName; }
```

C++ за 8 шагов (VIII)

- Обработка исключительных ситуаций

```
double div (double a, double b)
{
    if (b == 0.0)
    {
        throw 0.0;
    }
    return a / b;
}
```

```
try
{
  C = div (A, B);
}
catch(double v)
{
  //обработать деление на 0
  return;
}
```

ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ OPENFOAM

src/OpenFOAM/db/error/error.H

```
class error
:
    public std::exception,
    public messageStream
{

protected:

    // Protected data

        string functionName_;
        string sourceFileName_;
        label sourceFileLineNumber_;
```

Наследует от std::exception — поддерживает STL — следовательно и механизм try-catch

ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ OPENFOAM

Для генерации ошибки чаще всего используется макроопределение FoamFatalErrorIn

src/finiteVolume/lnInclude/clippedLinear.H

```
void calcWfLimit()
{
    if (cellSizeRatio_ <= 0 || cellSizeRatio_ > 1)
    {
        FatalErrorIn("clippedLinear::calcWfLimit()")
            << "Given cellSizeRatio of " << cellSizeRatio_
            << " is not between 0 and 1"
            << exit(FatalError);
    }

    wfLimit_ = cellSizeRatio_/(1.0 + cellSizeRatio_);
}
```

СПАСБО ЗА ВНИМАНИЕ!

$$pV = \nu RT$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \boldsymbol{U} = 0$$

```
fvm::ddt(rho) + fvc::div(phi)=0
```

$$\frac{\partial \rho \boldsymbol{U}}{\partial t} + \nabla \cdot (\rho \boldsymbol{U}\boldsymbol{U}) - \nabla \cdot \left(\mu \frac{1}{2} \left(\nabla \boldsymbol{U} + (\nabla \boldsymbol{U})^T \right) \right) = -\nabla p$$

```
fvm::ddt(rho, U) + fvm::div(phi,U) -
fvm::laplacian(mu,U)
=
-fvc::grad(p)
```