MATH 676

–

Finite element methods in scientific computing

Wolfgang Bangerth, Texas A&M University

Lecture 12:

A little bit of C++:

Templates

C++ templates

Templates are an elegant way in C++ to express similar things only once, even if they're not quite the same.

Our goal here:

Mathematical notation looks the same whether we are in 1d, 2d, or 3d.
Represent this invariance in our code!

Examples of math notation in 1d/2d/3d

Matrix assembly: $A_{ij} = \Sigma_K (\nabla \varphi_i, \nabla \varphi_j)_K = \Sigma_K \Sigma_{q=0}^{N_Q} \nabla \varphi_i(x_q^K) \cdot \nabla \varphi_j(x_q^K) w_q^K$

```
for (cell=begin; cell!=end; ++cell)
      apply d-dimensional quadrature formula
      to integrand on cell K
```

Simple error indicator: $\eta_K^2 = \frac{h}{24} \left\| [\boldsymbol{n} \cdot \nabla u_h] \right\|_{\partial K}^2$

```
for (all cells)
    for (all faces of this cell)
        apply (d-1)-dimensional quadrature formula
        to jump term on this face
```

Function templates in C++

Example:

```
template <typename number>
number sqr (const number x)   { return x*x; };
```

- *Not* a function
- But description of a *way to generate* one

Use:

```
double x,y;
x = sqr(y);
```

- Check for `sqr(double)`
- Check whether `sqr(double)` can be generated from a template, then compile it

Class templates in C++

Example:

```
template <typename number>
class Vector3 {
      number elements[3];
};
```

- *Not* a class
- But description of a *way to generate* a class

Use:

```
Vector3<double>     double_vector3;
```

- Creates class with 3 elements of type `double`
- Number/pos. of data known at compile time
- Compiler can optimize

Explicit specialization

Example:

```
template <>
class Vector3<bool> {
      int bit_field;
};
```

- This *is* a class

Use:

```
Vector3<double>     double_vector3;
Vector3<bool>       bool_vector3;
```

- First declaration generates class from template
- Second declaration takes explicitely specialized variant

Value templates in C++

Example:

```
template <unsigned int N>
class Vector {
      double elements[N];
};
```

- Class parameterized on number of elements

Use:

```
Vector<3> vector3;
```

- Generates class with three elements
- Number and position of data elements known at compile time; optimizations based on this information possible

Example of value templates in C++

```
template <unsigned int N>
double norm (const Vector<N> &v)
{
    double tmp = 0;
    for (unsigned int i=0; i<N; ++i)
        tmp += sqr(v.elements[i]);
    return sqrt(tmp);
};
```

- Actual value of *N* known at compile time
- Compiler will thus unroll loop
- Value templates faster than vectors of dynamic length

Value templates in deal.II – 1

Point in 'dim' space dimensions:

```
template <int dim>
class Point {
      double components[dim];
};
```

A 'structdim' dimensional object in 'spacedim' dimensional space:

```
template <int structdim, int spacedim>
class TriaObject {
      Point<spacedim>     vertices[1<<structdim];
      ...
      Point<spacedim>     vertex(unsigned int v);
};
```

Value templates in deal.II – 2

```
template <int spacedim>
class Cell   :   public TriaObject<spacedim,spacedim>
{
   Cell<spacedim> &
      neighbor (unsigned int neighbor_no);

   TriaObject<spacedim-1,spacedim> &
      face (unsigned int face_no);

   ...
};
```

“Traits”

```
template <>
class Triangulation<1> {
   typedef   TriaIterator<Cell<1> >              cell_iterator;
   typedef   void *                              face_iterator;
};


template <>
class Triangulation<2> {
   typedef   TriaIterator<Cell<2> >              cell_iterator;
   typedef   TriaIterator<TriaObject<1,2> >   face_iterator;
};
```

Examples of use – 1

Writing out the cells of a triangulation:

```
template <int dim>
void write_cells (Triangulation<dim> &tria)
{
   Triangulation<dim>::cell_iterator cell;
   for (cell=tria.begin(); cell!=tria.end(); ++cell)
      for (int v=0;
           v<GeometryInfo<dim>::vertices_per_cell;
           ++v)
        cout << cell->vertex(v) << endl;
};
```

Examples of use – 2

Compute error indicators on faces of cells:

```
template <int dim>
void ErrorIndicator<dim>::compute_indicators ()
{
  QGauss<dim-1> quadrature_formula(3);
  Triangulation<dim>::active_cell_iterator cell;
  for (cell=tria.begin(); cell!=tria.end(); ++cell)
     for (int f=0;
          f<GeometryInfo<dim>::faces_per_cell;
          ++f)
       integrate_on_face (cell->face(f),
                          quadrature_formula);
};
```

More on C++ templates

C++ templates are exceedingly helpful. But:

- Their syntax is clumsy and sometimes hard to read
- They often lead to very long error messages
- There is a surprising number of pitfalls

For more information, consult

- Your favorite C++ book
- The web:
  http://cplusplus.com/
  http://cplusplus.com/doc/tutorial/templates/
- The deal.II FAQ section on C++, linked to from
  http://www.dealii.org/

MATH 676

–

Finite element methods in scientific computing

Wolfgang Bangerth, Texas A&M University