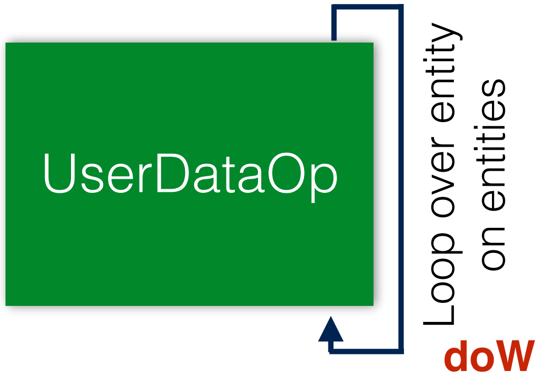
Type: OPROW or OPCOL (indices on rows or on columns)

Constructor:

```
UserDataOperator(const std::string &field_name, const
char type)
```



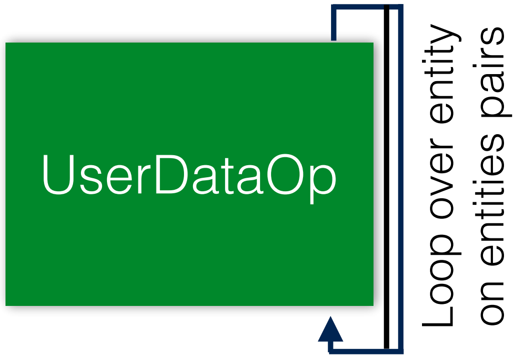
```
entities_set = { Vertices, Edge0, .., Edge5,
Face0, .., Face3, Volume }
for(o in operator_sequence)
  for(e in entities_set)
    o.doWork(side[e],type[e],ent_data[e])
```

doWork is overloaded method by user (loop is implicitly called by element)

Type: OPROWCOL (indices on rows & on columns)

Constructor:

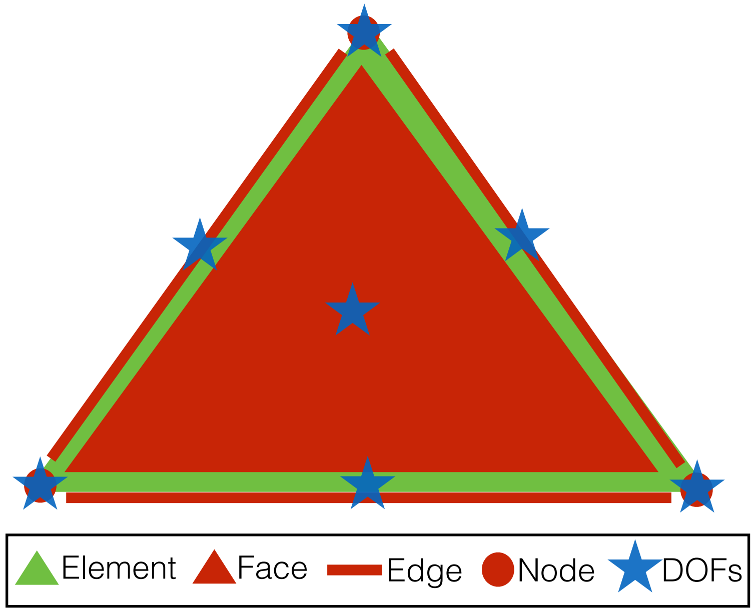
```
UserDataOperator(const std::string &row_field_name,
const std::string &col_field_name, const char type,
const bool symm=true)
```



```
entities_pair_set = { {Vertices, Vertices},
{Vertices,Edge0}, .., { Volume, Volume} }
for(o in operator_sequence)
 for(e in entities_pair_set)
  o.doWork(
   row_side[e.f],row_type[e.f],row_ent_data[e.f],
   col_side[e.s],col_type[e.s],col_ent_data[e.s]
  )
```

doWork is overloaded method by user (loop is implicitly called by element)

For square matrices & symmetric finite element *OPROW* & *OPCOL* are equivalent.
For *OPROWCOL*, when *symm = true*, only unique pairs are processed. It is third kind of operator, which not loop on entities of particular field, but entities of space, e.g. used to apply transformation to base functions. You can as well set *type = OPROW|OPROWCOL*.



Tetrahedral has base & DOFs on entities.

By space:

- **Space H1:** Vertices, 6 Edges, 4 Faces (Tri, Quad), 1 Volume
- **Space H-Curl:** 6 Edges,4 Faces, 1 Volume
- **Space H-Div:** 4 Faces, 1 Volume
- **Space L2:** 1 Volume (Tet, Prism, Hex, Wedge, …)

By order:

- **H1 order 1:** Only on Vertices
- **H1 order 2:** Vertices and Edges
- **H1 order 3:** Vertices, Edges & Faces
- **H1 order 4 and more:** Verices, Egdes, Faces and Volume

In similar way for other approximation spaces.

EntData:

- Values at DOFs
- Global/Local indices of DOFs
- Base/Space/Order/Sense
- Base functions & more