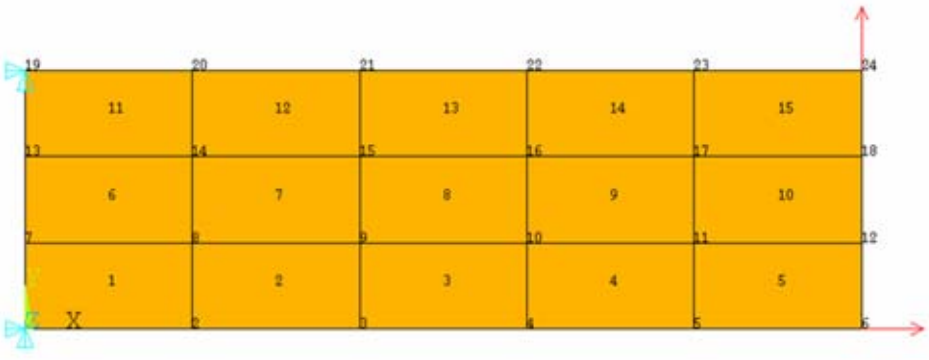
FELyX
Finite Element Library eXperiment

Use a parser to import FE Model

Write own parser

Solve structural FE Model




```
##############################
# This is a simple FE-Model
##############################
# NodeList
# NodeNb X Y Z
0 0.0000 0.00000 0.00000
1 20.0000 0.00000 0.00000
2 40.0000 0.00000 0.00000
3 60.0000 0.00000 0.00000
4 80.0000 0.00000 0.00000
5 100.000 0.00000 0.00000
6 0.00000 10.0000 0.00000
7 20.0000 10.0000 0.00000
8 40.0000 10.0000 0.00000
9 60.0000 10.0000 0.00000
10 80.0000 10.0000 0.00000
11 100.000 10.0000 0.00000
12 0.00000 20.0000 0.00000
13 20.0000 20.0000 0.00000
14 40.0000 20.0000 0.00000
15 60.0000 20.0000 0.00000
16 80.0000 20.0000 0.00000
…
…
…
#ElementList
#ElementNb Node1 Node2 Node3 Node4
0 0 1 7 6
1 1 2 8 7
2 2 3 9 8
3 3 4 10 9
4 4 5 11 10
5 6 7 13 12
6 7 8 14 13
7 8 9 15 14
…
…
…
```

Problem

- A txt-file with nodes and elements is given
- Use boost spirit library to import FE Model from a txt-file
  - Import txt-file in a buffer
  - Store nodes and elements in vectors
  - Introduce loadcase
  - Solve
  - Write results into a txt-file
- Same example as the previous
  - One new function is implemented

```
void MyStructObject::create_model_from_file(std::string fname_, std::string fpath_)
```

  - The other things are similar to the last example “Build your own FE Model completely in FELyX”

- Useful link
  - http://www.boost.org/libs/spirit/index.html

Format of Text File

- Text file includes nodes and elements, but no material and loadcase

```
############################
# This is a simple FE-Model
############################
# NodeList
# NodeNb X Y Z
0 0.0000 0.00000 0.00000
1 20.0000 0.00000 0.00000
2 40.0000 0.00000 0.00000
3 60.0000 0.00000 0.00000
4 80.0000 0.00000 0.00000
5 100.000 0.00000 0.00000
6 0.00000 10.0000 0.00000
7 20.0000 10.0000 0.00000
8 40.0000 10.0000 0.00000
9 60.0000 10.0000 0.00000
10 80.0000 10.0000 0.00000
11 100.000 10.0000 0.00000
12 0.00000 20.0000 0.00000
13 20.0000 20.0000 0.00000
14 40.0000 20.0000 0.00000
15 60.0000 20.0000 0.00000
16 80.0000 20.0000 0.00000
17 100.000 20.0000 0.00000
18 0.00000 30.0000 0.00000
19 20.0000 30.0000 0.00000
20 40.0000 30.0000 0.00000
21 60.0000 30.0000 0.00000
22 80.0000 30.0000 0.00000
23 100.000 30.0000 0.00000
#ElementList
#ElementNb Node1 Node2 Node3 Node4
0 0 1 7 6
1 1 2 8 7
2 2 3 9 8
3 3 4 10 9
4 4 5 11 10
5 6 7 13 12
6 7 8 14 13
7 8 9 15 14
8 9 10 16 15
9 10 11 17 16
10 12 13 19 18
11 13 14 20 19
12 14 15 21 20
13 15 16 22 21
14 16 17 23 22
```

Node List with:
Node number
Node coordinates x, y and z

Element List with:
Element number
Nodes of the element

Parse txt-file

```
void MyStructObject::create_model_from_file(std::string fname_, std::string fpath_)
{
  using namespace boost::spirit;

  typedef char * iterator_t;
  typedef scanner<iterator_t> scanner_t;
  typedef rule<scanner_t> rule_t;

  std::string filepath((fpath_ + "/" + fname_).c_str());
  std::ifstream infile(filepath.c_str());

  if (!infile.good())
  {
    std::cout << "ERROR: In ReadFileToMemory: Could not open the file " << std::endl;
      }

  infile.seekg(0,std::ifstream::end);
  unsigned long filesize=infile.tellg();
  infile.seekg(0);

  char * buffer;    // the buffer, where the text file is stored for fast parsing
  buffer = new char [filesize];

  infile.read(buffer,filesize);
  infile.close();

  std::vector<double> x_vec(0), y_vec(0), z_vec(0);
  std::vector<unsigned> nodeIndex, elementIndex, firstNode, secondNode, thirdNode,
fourthNode;

  rule_t read_coordinates_line
  = uint_p[ push_back_a( nodeIndex ) ] >> blank_p >>
    real_p[ push_back_a( x_vec ) ] >> blank_p >>
    real_p[ push_back_a( y_vec ) ] >> blank_p >>
    real_p[ push_back_a( z_vec ) ] >> eol_p;

  BOOST_SPIRIT_DEBUG_RULE( read_coordinates_line );

  rule_t read_elements_line
  = uint_p[ push_back_a( elementIndex ) ] >> blank_p >>
    uint_p[ push_back_a( firstNode ) ] >> blank_p >>
    uint_p[ push_back_a( secondNode ) ] >> blank_p >>
    uint_p[ push_back_a( thirdNode ) ] >> blank_p >>
    uint_p[ push_back_a( fourthNode ) ] >> eol_p;
  BOOST_SPIRIT_DEBUG_NODE( read_elements_line );

  rule_t comment_parser = comment_p('#'); //ch_p( '#' ) >> *( anychar_p - eol_p ) >>
eol_p ;
  BOOST_SPIRIT_DEBUG_NODE( comment_parser );

  rule_t read_any_line = read_coordinates_line | read_elements_line | comment_parser ;
  BOOST_SPIRIT_DEBUG_NODE( read_any_line );

  iterator_t first( buffer );
  iterator_t last = first + filesize;
  unsigned long psize = parse( first,last, *(read_coordinates_line | read_elements_line |
comment_parser |eol_p ) ).length;

  if ( psize < filesize ) {
    std::cout <<  "WARNING: Not the entire input file " << filepath << " has been parsed."
<< std::endl;
    std::cout <<  "Only " << psize << " characters of total " << filesize << " are read" <<
std::endl;
  }
```

Define types of SPIRIT constructs

The file path

Get size of file

Allocate memory for file content

Read content of infile to buffer

Close the infilestream, it is no longer used

Here goes the rule to read a single line consisting of 3 comma delimited real numbers

```
  Materials.push_back( new fe_base::IsotropicMaterial() );
  Materials[ 0 ] ->Set( "E", 70000 );
  Materials[ 0 ] ->Set( "nu", 0.3 );

  Nodes.resize( 24 );

  for ( unsigned i = 0; i < 24 ; ++i )
   {
      Nodes[nodeIndex[i]].set( x_vec[i], y_vec[i], z_vec[i] );
   }


  Resize element ptrvector
  Elements.resize( firstNode.size() );

  PtrVector<element_type*>::iterator ele_it = Elements.begin();

  for ( unsigned i = 0; i < Elements.size() ; ++i )
   {
     ( *ele_it ) = new fe_base::Plane182;

     ( *ele_it ) ->SetNodeIter( 0, Nodes.begin() + firstNode[i] );
     ( *ele_it ) ->SetNodeIter( 1, Nodes.begin() + secondNode[i] );
     ( *ele_it ) ->SetNodeIter( 2, Nodes.begin() + thirdNode[i] );
     ( *ele_it ) ->SetNodeIter( 3, Nodes.begin() + fourthNode[i] );

     ( *ele_it ) ->SetMaterialPtr( Materials[ 0 ] );

     ++ele_it;

   }

  BoundaryConditions.resize( 3 );

  BoundaryConditions[ 0 ].set( Dx, 0.0 );
  BoundaryConditions[ 0 ].set( Dy, 0.0 );

  Nodes[ 0 ].set( &BoundaryConditions[ 0 ] );
  Nodes[ 18 ].set( &BoundaryConditions[ 0 ] );
  BoundaryConditions[ 1 ].set( Fx, 100.0 );
  BoundaryConditions[ 2 ].set( Fy, 100.0 );
  Nodes[ 5 ].set( &BoundaryConditions[ 1 ] );
  Nodes[ 23 ].set( &BoundaryConditions[ 2 ] );

  delete[] buffer;
}
```

Create new Material

Fill the nodes of the model

Resize node vector

Fill in coordinates of nodes

Create elements of model

We need a element container iterator

Fill in the nodes of the elements

Create new plane element

Add node iterator to plane elements

Set material

Create and assign boundary conditions

Free the memory allocated to buffer