NOTES FROM WEEK 49-51

In Week 49 -51, work has been carried out to:

1. Improve the structure of input file *input.iron* in order to make it align with the structure the *FortranExample.f90* file
2. Parsing algorithm is reformulated as per the new structure of the *input.iron* file.
3. *FortranExample.f90* file is modified with an intention to:
4. Introduce allocatable data structures with derived types in *FortranExample.f90* file
5. Assign the field UserNumbers to their respective Regions(/mesh/EquationSets etc.) using do loops.
6. Make it generic enough to solve laplace and finite elasticity problems.

The comprehensive explanation of the above enumerated work is as follow.

1-Input File

The input file consist of the following blocks in any arbitrary order.

1. Region
2. CoordinateSystem
3. BasisType
4. MeshFeature
5. EquationSet
6. ProblemType
7. DependentField
8. MaterialField
9. FiberField
10. GeometricField
11. ControlLoop

12. SolverSetting
13. BoundaryConditions

Moreover each block consist of a number of parameters used to define the block. The parameters are enumerated in table 1.

Block	**Parameters**
EquationsSet	EquationsSet_ID Class Type Subtype
Problem	Problem_ID Class Type Subtype
Region	Region_ID (more features added later)
MaterialField	MaterialField_ID MaterialField_Parameters
FiberField	FiberField_ID FiberField_Parameters
GeometricField	GeometricField_ID (more features added later)
ControlLoop	ControlLoop_ID Type Increments
SolverSettings	Solver_ID EquationSet _Solver_Type Preconditioner EquationSolverTolerance NewtonMaximumIterations NewtonTolerance
CoordinateSystem	CoordinateSystem_ID CoordinateSystem_Type
Basis	Basis_ID Basis_Type PressureBasis_Type
BoundaryConditions	BoundaryConditions_ID Dirichelet Neumann_CF (tractions) Neumann_Pressure

2-Syntax of the input file

Following is the syntax that should be taken into account in order to make a input.iron file readable to the parsing algorithm.
2.1 Each block starts with keyword START_ // block name and ends with keyword END // block name. For example the block used to define the Region starts as Start_Region and ends as End_Region. Note that all keywords in *input.iron* file are case insensitive.
2.2 A blank line act as a separator between two parameters of a block.

```
    CLASS
EQUATIONS_SET_ELASTICITY_CLASS
[one or multiple blank lines]
TYPE
EQUATIONS_SET_FINITE_ELASTICITY_TYPE
```

2.3 Indentation is allowed.
2.4 Comments start with exclamation mark symbol!
2.5 Comments can be used on right hand side of the keywords as well. For instance

```
    [Keyword]. [Comment]
MAXIMUM ITERATIONS !! Wont be utilized in case the SOLVER TYPE is direct.
```

3-Search Algorithm

The search algorithm initializes variables to the number of time each blocks is defined in the input file and later on these parmeters are used in the FortranExampleFile.f90 to allocate the size of the data structures. This task is performed in the file *AllocatingDerivedDataStructures.f90.*.
The variables are as follow:

```
    INTEGER(CMISSIntg) :: num_of_PressureBasis
INTEGER(CMISSIntg) :: num_of_Basis
INTEGER(CMISSIntg) :: num_of_BoundaryCondition
INTEGER(CMISSIntg) :: num_of_WorldCoordinateSystem
INTEGER(CMISSIntg) :: num_of_CoordinateSystem
INTEGER(CMISSIntg) :: num_of_Mesh
INTEGER(CMISSIntg) :: num_of_Decomposition
INTEGER(CMISSIntg) :: num_of_Equation
INTEGER(CMISSIntg) :: num_of_EquationsSet
INTEGER(CMISSIntg) :: num_of_GeometricField
INTEGER(CMISSIntg) :: num_of_FiberField
INTEGER(CMISSIntg) :: num_of_MaterialField
INTEGER(CMISSIntg) :: num_of_DependentField
INTEGER(CMISSIntg) :: num_of_EquationSetField
INTEGER(CMISSIntg) :: num_of_Field
```

```
INTEGER(CMISSIntg) :: num_of_Problem
INTEGER(CMISSIntg) :: num_of_Region
INTEGER(CMISSIntg) :: num_of_WorldRegion
INTEGER(CMISSIntg) :: num_of_Solver
INTEGER(CMISSIntg) :: num_of_LinearSolver
INTEGER(CMISSIntg) :: num_of_SolverEquations
INTEGER(CMISSIntg) :: num_of_ControlLoop
INTEGER(CMISSIntg) :: num_of_GeneratedMesh
```

4-Modifications in *FortranExample.90* file

The following modifications are introduced in the *FortranExample.90* file.

1. Allocatable data structures with derived types are introduced in file Derived-Types.f90. A glimpse of it is shown as follows.

```
      !CMISS variables
 type basis_array
      type(cmfe_BasisType), allocatable :: Basis(:),PressureBasis(:)
end type basis_array
 type boundary_conditions_array
      TYPE(cmfe_BoundaryConditionsType), allocatable :: BoundaryConditions(:)
end type boundary_conditions_array
 type coordinate_system_array
      type(cmfe_CoordinateSystemType) , allocatable:: CoordinateSystem(:), WorldCoordinateSystem(:)
end type coordinate_system_array
 type mesh_array
      type(cmfe_MeshType), allocatable:: Mesh(:)
end type mesh_array
 type decomposition_array
      type(cmfe_DecompositionType) , allocatable :: Decomposition(:)
end type decomposition_array
 type equations_array
      type(cmfe_EquationsType) , allocatable :: Equations(:)
end type equations_array
 type equations_set_array
      type(cmfe_EquationsSetType) , allocatable :: EquationsSet(:)
end type equations_set_array
 type field_type_array
      type(cmfe_FieldType), allocatable :: GeometricField (:), FibreField (:), MaterialField (:), EquationsSetField(:)
end type field_type_array
 type fields_type_array
      type(cmfe_FieldsType) , allocatable :: Fields(:)
end type fields_type_array
 type problem_type
      type(cmfe_ProblemType) , allocatable :: Problem(:)
end type problem_type
 type region_type
      TYPE(cmfe_RegionType) , allocatable :: Region(:),WorldRegion(:)
end type region_type
 type solver_type
      TYPE(cmfe_SolverType) , allocatable:: Solver(:),LinearSolver(:)
end type solver_type
 type solvers_equations_type
      TYPE(cmfe_SolverEquationsType) , allocatable :: SolverEquations(:)
end type solvers_equations_type
 type control_loop_type
      TYPE(cmfe_ControlLoopType) , allocatable:: ControlLoop(:)
end type control_loop_type
 type generate_mesh_type
      TYPE(cmfe_GeneratedMeshType) , allocatable:: GeneratedMesh(:)
end type generate_mesh_type
 type dependent_field_type
      TYPE(cmfe_FieldType) , allocatable:: DependentField(:)
end type dependent_field_type
 TYPE(basis_array)
:: all_Basis, all_PressureBasis
```

```
TYPE(boundary_conditions_array) :: all_BoundaryConditions
TYPE(coordinate_system_array) :: all_CoordinateSystem
TYPE(coordinate_system_array) :: all_WorldCoordinateSystem
TYPE(mesh_array) :: all_Mesh
TYPE(decomposition_array) :: all_Decomposition
TYPE(equations_array) :: all_Equations
TYPE(equations_set_array) :: all_EquationsSet
TYPE(field_type_array) :: all_GeometricField
TYPE(field_type_array) :: all_FibreField
TYPE(field_type_array) :: all_MaterialField
TYPE(field_type_array) :: all_EquationsSetField
TYPE(fields_type_array) :: all_Fields
TYPE(problem_type) :: all_problem
TYPE(region_type) :: all_Region
TYPE(region_type) :: all_WorldRegion
TYPE(solver_type) :: all_Solver
TYPE(solver_type) :: all_LinearSolver
TYPE(solvers_equations_type) :: all_SolverEquations
TYPE(control_loop_type) :: all_ControlLoop
TYPE(generate_mesh_type) :: all_GeneratedMesh
TYPE(dependent_field_type) :: all_DependentField
```

2- The above mentioned derived data structures are used allocated using the variables initialized by the search algorithm (in section 3). This task has been prefomed in the file *AllocatingDerivedDataStructures.f90*. S glimpse of the file is shown as follows.

```
    allocate(all_Basis%Basis(num_of_Basis))
allocate(all_PressureBasis%PressureBasis(num_of_PressureBasis))
allocate(all_BoundaryConditions%BoundaryConditions(num_of_BoundaryCondition))
allocate(all_CoordinateSystem%CoordinateSystem(num_of_CoordinateSystem))
allocate(all_WorldCoordinateSystem%WorldCoordinateSystem(num_of_CoordinateSystem))
allocate(all_Mesh%Mesh(num_of_Mesh))
allocate(all_Decomposition%Decomposition(1)) allocate(all_Equations%Equations(num_of_EquationsSet))
allocate(all_EquationsSet%EquationsSet(num_of_EquationsSet))
allocate(all_GeometricField%GeometricField(num_of_GeometricField))
allocate(all_FibreField%FibreField(num_of_FiberField))
allocate(all_MaterialField%MaterialField(num_of_MaterialField))
allocate(all_EquationsSetField%EquationsSetField(num_of_EquatioSet))
allocate(all_Fields%Fields(num_of_Fields))
allocate(all_Problem%Problem(num_of_problems))
allocate(all_Region%Region(num_of_Regions))
allocate(all_WorldRegion%WorldRegion(num_of_Regions))
allocate(all_Solver%Solver(num_of_Solver))
allocate(all_LinearSolver%LinearSolver(num_of_Solver))
allocate(all_SolverEquations%SolverEquations(1))
allocate(all_ControlLoop%ControlLoop(num_of_ControlLoop))
allocate(all_GeneratedMesh%GeneratedMesh(num_of_Mesh))
allocate(all_DependentField%DependentField(num_of_DependentField))
allocate (CoordinateSystemUserNumber(num_of_CoordinateSystem))
allocate (RegionUserNumber(num_of_Region))
allocate (BasisUserNumber(num_of_Basis))
allocate (PressureBasisUserNumber(num_of_PressureBasis))
allocate (GeneratedMeshUserNumber(num_of_GeneratedMesh))
allocate (MeshUserNumber(num_of_Mesh)) allocate
(DecompositionUserNumber(num_of_Decomposition))
allocate (FieldGeometryUserNumber(num_of_GeometricField))
allocate (FieldFibreUserNumber(num_of_FiberField))
allocate (FieldMaterialUserNumber(num_of_MaterialField))
allocate (FieldDependentUserNumber(num_of_DependentField))
allocate (EquationSetUserNumber(num_of_EquationsSet))
allocate (EquationsSetFieldUserNumber(num_of_EquationSetField ))
allocate (ProblemUserNumber(num_of_Problem))
 !!!!!!!!!!!!!!!!!! Assigning tags. NOTE: Each tag should be distinct !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! j = 1
do i = 1, num_of_CoordinateSystem
    CoordinateSystemUserNumber(num_of_CoordinateSystem) = j j = j + 1
end do
do i = 1, num_of_Region
    RegionUserNumber(num_of_Region) = j j = j + 1
end do
do i = 1, num_of_Basis
```

```
      BasisUserNumber(num_of_Basis) = j j = j + 1
end do
do i = 1, num_of_PressureBasis
      PressureBasisUserNumber(num_of_PressureBasis) = j
      j = j + 1
end do
do i = 1,num_of_GeneratedMesh
      GeneratedMeshUserNumber(num_of_GeneratedMesh) = j
j = j + 1
end do
do i = 1, num_of_Mesh
      MeshUserNumber(num_of_Mesh) = j
j = j + 1
end do
do i = 1, num_of_Decomposition
      DecompositionUserNumber(num_of_Decomposition) = j
j = j + 1
end do
do i = 1, num_of_GeometricField
      FieldGeometryUserNumber(num_of_GeometricField) = j
j = j + 1
end do
do i = 1, num_of_FiberField
      FieldFibreUserNumber(num_of_FiberField) = j
j = j + 1
end do
do i = 1, num_of_MaterialField
      FieldMaterialUserNumber(num_of_MaterialField) = j
j = j + 1
end do
do i = 1, num_of_DependentField
      FieldDependentUserNumber(num_of_DependentField) = j
j = j + 1
end do
do i = 1, num_of_EquationsSet
      EquationSetUserNumber(num_of_EquationsSet) = j
j = j + 1
end do

      do i = 1, num_of_EquationSetField
      EquationsSetFieldUserNumber(num_of_EquationSetField) = j
j = j + 1
end do
do i = 1, num_of_Problem
      ProblemUserNumber(num_of_Problem) = j
j = j + 1
end do
```

3- The data structures with derived types allocated above is used further on in the fortran file insuch as way that every parameter defined in the input file is linked to its respective Mesh/Region/EquationSet using do loops. For instance in the following case the coordinate system defined in the input file (with CoordinateSystem_ID: Solid) is assigned to its respective region region (with Region_ID: Solid).

```
      DO i = 1,num_of_CoordinateSystem !!!!!!!!!!!!!  introducing a LOOP for assigning coordinate systems to their respective
regions
!Initialize cmiss
CALL cmfe_Initialise(all_WorldCoordinateSystem%WorldCoordinateSystem(i),all_WorldRegion%WorldRegion(i),Err)
CALL cmfe_ErrorHandlingModeSet(CMFE_ERRORS_TRAP_ERROR,Err)
!Set all diganostic levels on for testing
!CALL cmfe_DiagnosticsSetOn(CMFE_FROM_DIAG_TYPE,[1,2,3,4,5],"Diagnostics",["PROBLEM_RESIDUAL_EVALUATE"],Err)
!Get the number of computational nodes and this computational node number
CALL cmfe_ComputationalNumberOfNodesGet(NumberOfComputationalNodes,Err)
CALL cmfe_ComputationalNodeNumberGet(ComputationalNodeNumber, Err)
NumberGlobalXElements=str2int(Mesh_arg4(1,i))
NumberGlobalYElements=str2int(Mesh_arg4(2,i))
NumberGlobalZElements=str2int(Mesh_arg4(3,i))
NumberOfDomains=NumberOfComputationalNodes
!Create a 3D rectangular cartesian coordinate system
CALL cmfe_CoordinateSystem_Initialise(all_CoordinateSystem%CoordinateSystem(i),Err)
```

```
CALL cmfe_CoordinateSystem_CreateStart(CoordinateSystemUserNumber(i),all_CoordinateSystem%CoordinateSystem(i),Err)
Call cmfe_CoordinateSystem_TypeSet(CoordinateSystemUserNumber(i), and
match_coordinate_system(CoordinateSystem_arguments(2,i)), err)
CALL cmfe_CoordinateSystem_CreateFinish(all_CoordinateSystem%CoordinateSystem(i),Err)
!Create a region and assign the coordinate system to the region
CALL cmfe_Region_Initialise(all_Region%Region(i),Err)
CALL cmfe_Region_CreateStart(RegionUserNumber(i),all_WorldRegion%WorldRegion(i),all_Region%Region(i),Err)
CALL cmfe_Region_LabelSet(all_Region%Region(i),"Region",Err)
CALL cmfe_Region_CoordinateSystemSet(all_Region%Region(i),(all_CoordinateSystem%CoordinateSystem(i)),Err)
CALL cmfe_Region_CreateFinish(all_Region%Region(i),Err)
END DO
```

5-Suggestions and questions to be discussed in the next meeting

Following are some suggestions and questions to be discussed in the next meeting.

1. Since the large displacement case study was quite trivial in terms of the number of regions, mesh , problems etc (the magic number is always 1) , therefore I cannot rigorously check how robust my logic is for let say linking basis to their respective mesh or coordinate system to its respective region. Therefore a case study involving multiple domains (meshes/regions etc.) needs to be solved with the developed parsing algorithm to verify this aspect.

2. A block of code is introduced in the *FortranExample.f90* file to implement pressure and traction Neumann is as follow. This way of implementing pressure BC is used after examining the documentation of OPENCMISS. Does it seem alright to you?

```
    DO NN=1,SIZE(SurfaceNodes,1)
NODE=SurfaceNodes(NN)
CALL cmfe_Decomposition_NodeDomainGet(Decomposition,NODE,1,NodeDomain,Err)
IF(NodeDomain==ComputationalNodeNumber) THEN
CALL cmfe_BoundaryConditions_SetNode(BoundaryConditions,DependentField,
CMFE_FIELD_DELUDELN_VARIABLE_TYPE,1,1,NODE,
ABS(NormalXi), CMFE_BOUNDARY_CONDITION_PRESSURE,2,Err)
ENDIF
ENDDO
```

Goals for the upcoming week In Week 1 and 2,

1. The formulated FortranExample.f90 file will be generalized and verified from Laplace problem. In my opinion the formulated FortranExample.f90 with few more modification can be used to run the lapalcian problem.

2. Secondly a case study will be solved involving multiple domains (meshes /regions etc.) with the developed parsing algorithm and the fortranExample.90 file with intentions highlighted in bullet point 1 of section 4.