Developing a parsing algorithm for OPENCMISS to setup a generic simulation based on input file

Presenter: Waleed Mirza
Contributors: Andreas Hessenthaler and Waleed Mirza

Motivation / Advantages

The developed algorithm will allow user to,

- Set up a generic simulation in OPENCMISS by providing instructions in an input file *(just like .inp file in Abaqus or .ans file in Ansys).*
- To change the simulation parameters without recompiling the code.

Road Map Of Presentation

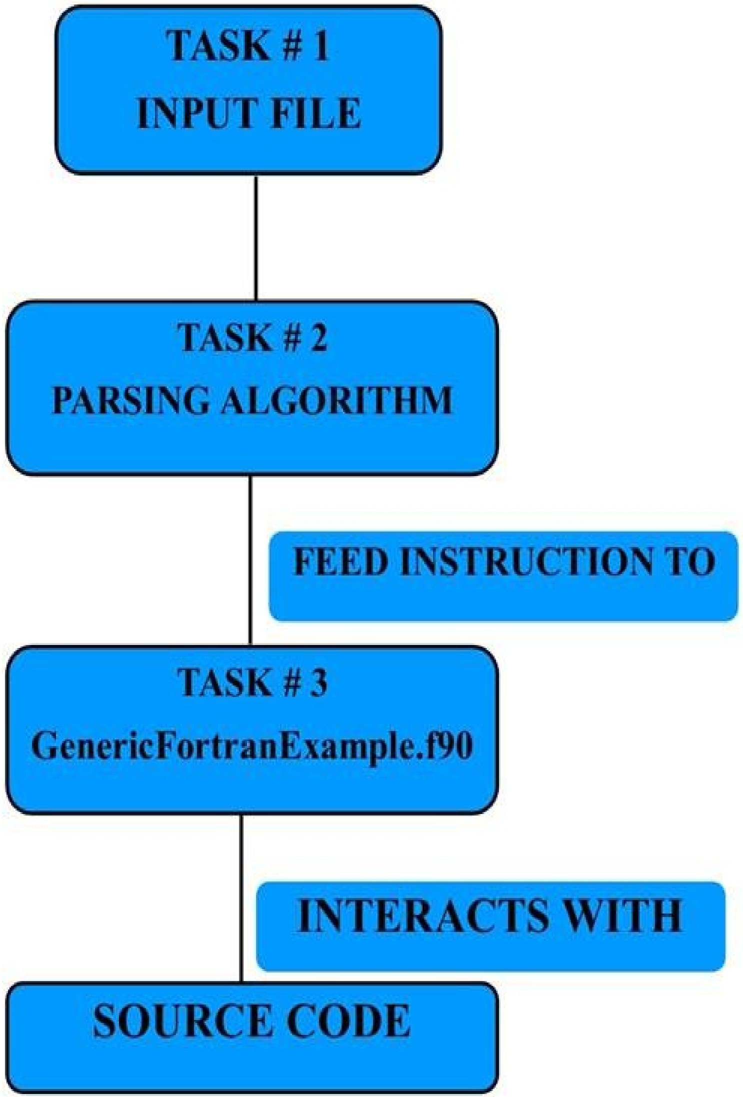
- Overview of the project
- How to execute an input file from Bash terminal ?
- Syntax and functionalities of input file.
- Different aspects of parsing algorithm
- Case studies
- Future work

Project overview

1- Develop layout/structure for the input file.

2- Develop a *GenericFortranExample.f90* file capable of setting up a generic simulation.

3- Develop a parsing algorithm that can pick input instructions from the input file and feeds them in the *GenericFortranExample.f90* file



Executing input file from bash

Given a binary file (for instance ***GenericCaseStudy.exe***) and the input file (*for instance* ***input.iron***) , a simulation can be executed by providing absolute path of the input file as a command line argument.

- $ <Absolute path to the binary file >/ *GenericCaseStudy* *<Absolute path to the input file file >/ input.iron*
-
-
-

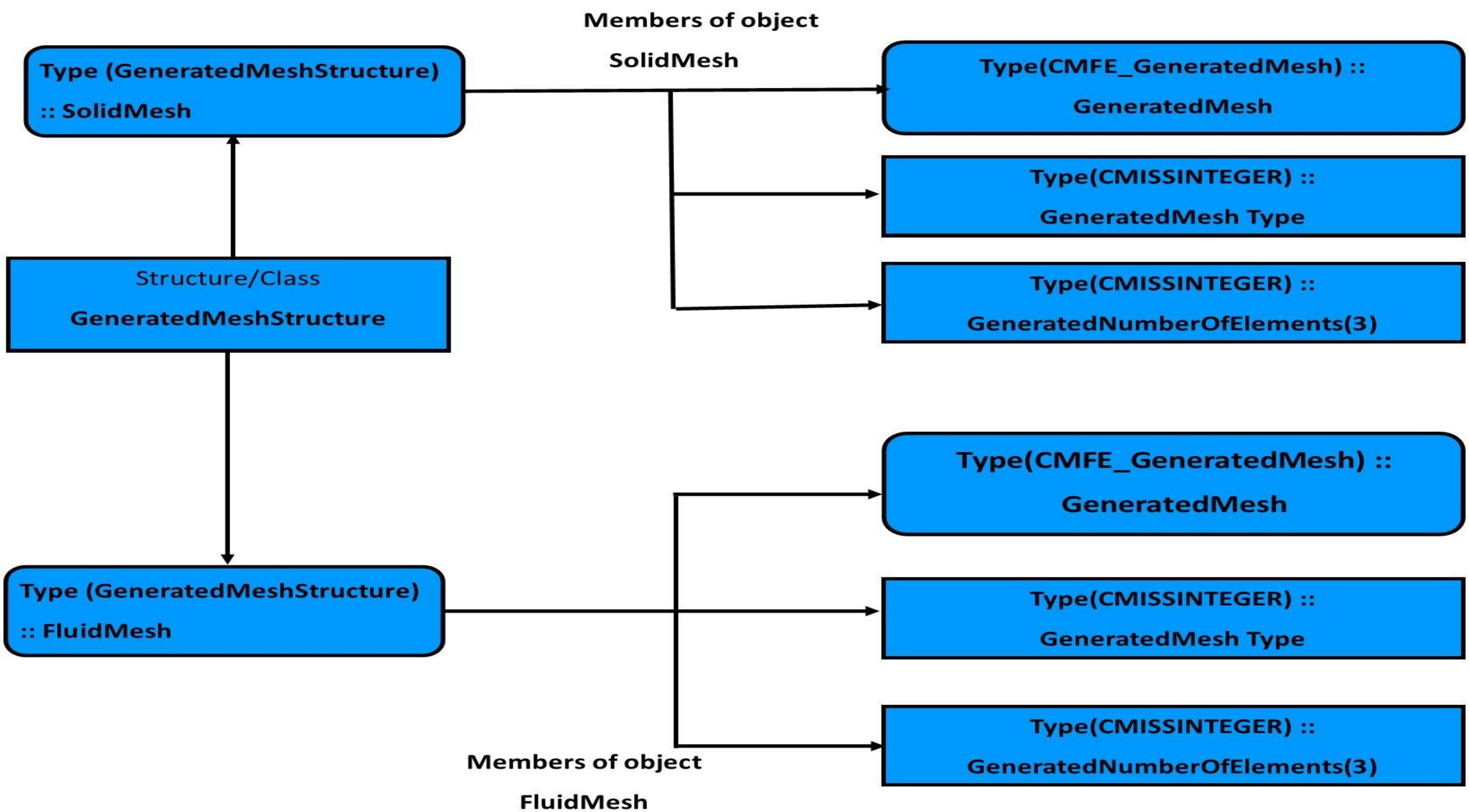
1- Preliminary Concepts

1) Structure of OPENCMISS involves objects of different derived types. Each derived type object contains different simulation parameters. For instance:
   Objects of *CMFE_GeneratedMesh* Type contain information of mesh parameters such as mesh size , topology etc.

2) Objects of *CMFE_BoundaryConditions* Type contains information of the boundary conditions such as location of BCs, prescribed values etc.

3) With instructions provided in the input file, these derived types objects are created and appropriate information is stored in them.

1- Preliminary Concepts (continued)



2- Syntax of Input file

1- The input file is divided in a **18 blocks** and each block generates a derived type object and contains information which is later stored in different members of the object.

- BASIS block
- GENERATED_MESH Block
- BOUNDARY_CONDITION Block
- CONTROL_LOOP block
- MATERIAL_FIELD block
- DEPENDENT_FIELD block etc.

2- Syntax of Input file

2- Each block starts and ends with the keywords *START_<BLOCK NAME >*and *END_<BLOCK NAME>* respectively.

3- Each block encapsulate set of input arguments.

1.

```
START_BASIS

BASIS_ID
FLUID

NumberOfGaussXi      ! For numerical integration
3   ,   3

BASIS_INTERPOLATION_TYPE
LINEAR_LAGRANGE_INTERPOLATION

END_BASIS
```

2- Syntax of Input file

4- The syntax of input is case insensitive. But users are encouraged to write information in upper case.

5- Comments should be started with exclamation mark *!*

6- Blank line can be introduced by the user in the input file for readability convenience.

7- The input file should be finished with the keyword

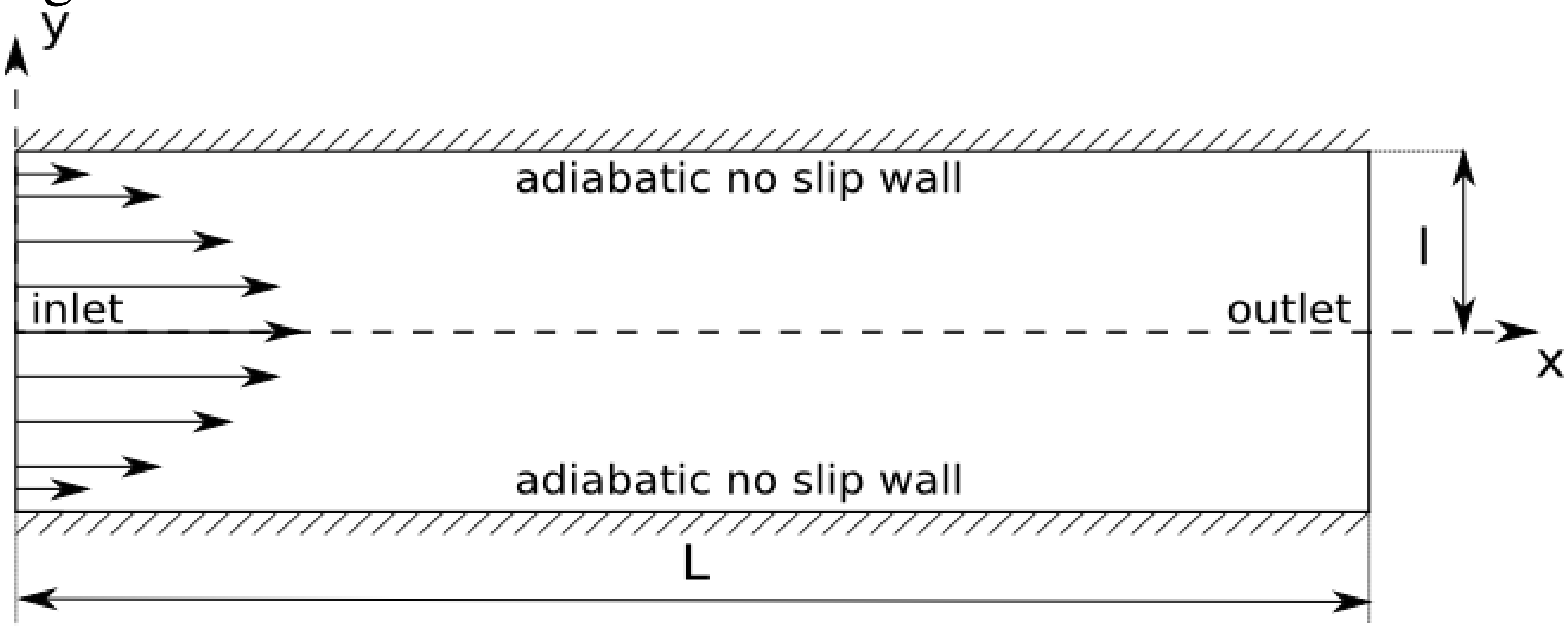
STOP_PARSING

8- Please be careful with the spellings .

2- Syntax of Input file (Continued)

Example of an input file

Here an input file of the case study involving 2D fluid flow in a rectangular channel will be shown.



At this point I will pull out the accompanying input file and explain it line by line.

3- Parsing Algorithm

3.1- Nomenclature of data structures

The nomenclature of the data structures have been established with an aim to make them **self descriptive** and **clear in terms of readability.**

For instance in

all_GeneratedMesh(:)%GeneratedMeshInterpolationType

1- ***all_GeneratedMesh(:)*** is an array of objects of generated mesh.

Size of the array = number of times the GENERATED_MESH block defined in the input file. For example for an FSI study there will be three GENERATED_MESH blocks i.e. for SOLID , FLUID and INTERFACE.”

3.1- Nomenclature of data structures

2- ***GeneratedMeshInterpolationType*** is a string type member of object ***all_GeneratedMesh(:)***. So in an FSI study each domain (Solid, Fluid and Interface) can have a different Interpolation type

all_GeneratedMesh(1)%GeneratedMeshInterpolationType = "QUADRATIC_LAGRANGE_INTERPOLATION"

all_GeneratedMesh(2)%GeneratedMeshInterpolationType = "QUADRATIC_LAGRANGE_INTERPOLATION"
all_GeneratedMesh(3)%GeneratedMeshInterpolationType = "LINEAR_LAGRANGE_INTERPOLATION"

3.1- Nomenclature Of Data Structures(Conti.)

Question: In a multidomain problem how will the algorithm recognize which interpolation type belongs to which domain ????

Answer: Block Ids will help algorithm with that.

all_GeneratedMesh(1)%GeneratedMeshId = "SOILD"

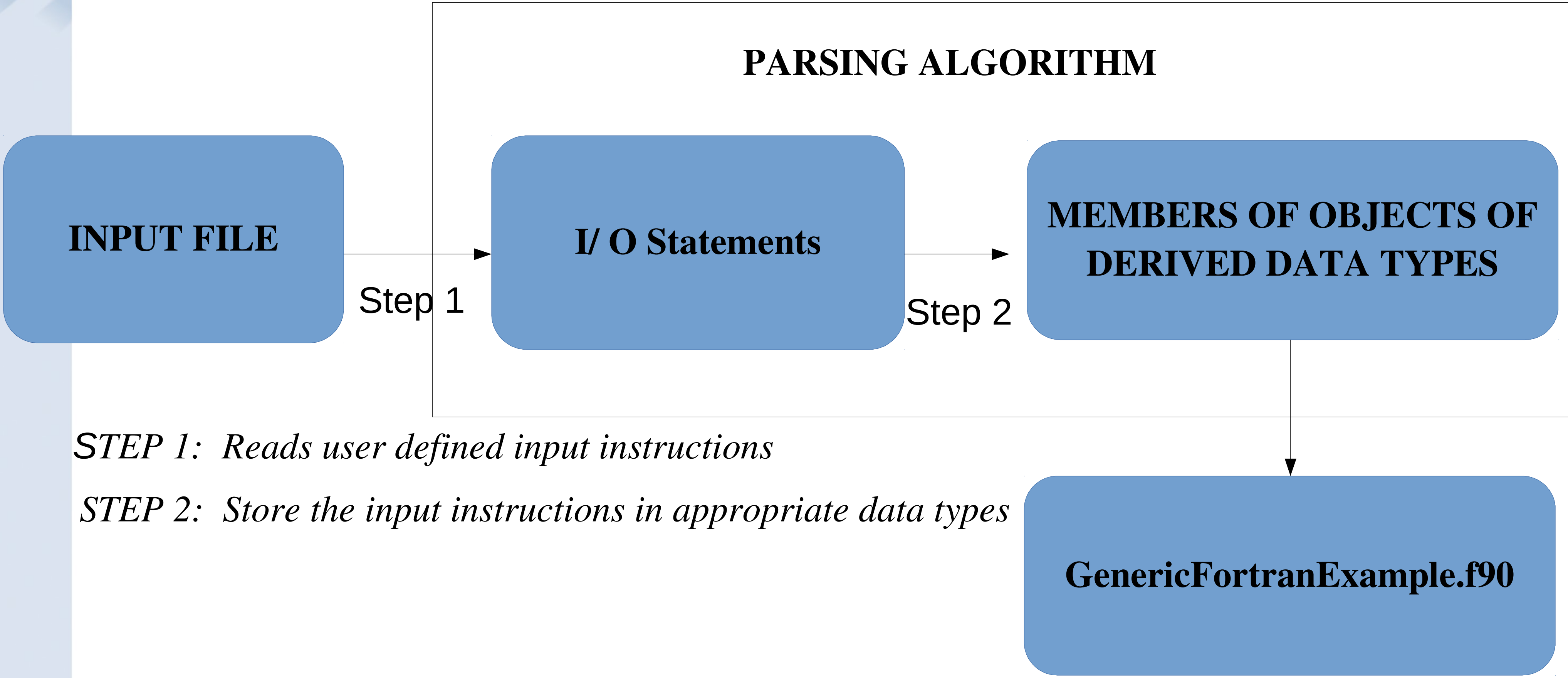
all_GeneratedMesh(2)%GeneratedMeshId = "FLUID"

all_GeneratedMesh(3)%GeneratedMeshId = "INTERFACE"

Section 4 will explain this feature of the algorithm more in detail.

3- Parsing Algorithm (Conti.)

In a Nutshell :



STEP 1: Reads user defined input instructions

STEP 2: Store the input instructions in appropriate data types

4- GenericFortranExampleFile.f90

- This is the file where simulation is setup using the parameters define in the input file.
- As of now, the *GenericFortranExampleFile.f90* is evolved enough to setup laplace , fluid and solid mechanics problems.
- Nevertheless, *GenericFortranExampleFile.f90* is quite amenable and modifiable for simulations of other classes.

4- GenericFortranExampleFile.f90

- Objects of different data type are linked here. For instance
- Objects of *all_Region(:)%Region* and *all_CoordinateSystem(:)%CoordinateSystem* with similar data types are linked together i.e.
- CALL cmfe_Region_CoordinateSystemSet(*all_Region(RegionIdx)%Region*, & *all_CoordinateSystem(CoordinateSystemIdx)%CoordinateSystem*,Err)
- The Algorithm has to make sure in the subroutine above , *CoordinateSystemIdx* and *RegionIdx* are such that
- *all_CoordinateSystem(CoordinateSystemIdx)%CordinateSystemId* = *"SOLID"*
- *all_Region(RegionIdx)%RegionId* = *"SOLID"*

4- GenericFortranExampleFile.f90

- Thats where the ***MATCH_ID()*** subroutine kicks in
- call subroutine MATCH_ID(all_Region(RegionIdx)%RegionId, &
  & all_CoordinateSystem(:)%CoordinateSystemId,
  & CoordinateSystemIdx)

all_Region(RegionIdx)
%RegionId = “FLUID”

all_Csys(1)
%CsysId = “SOLID”

all_Csys(2)
%CsysId = “FLUID”

all_Csys(3)
%CsysId =
“INTERFACE”

Answer:
CoordinateSystemIdx = 2

4- Case Studies # 1
4.1 Laplace Problem

Governing equation to be solved

$$\nabla^2 \varphi = 0$$

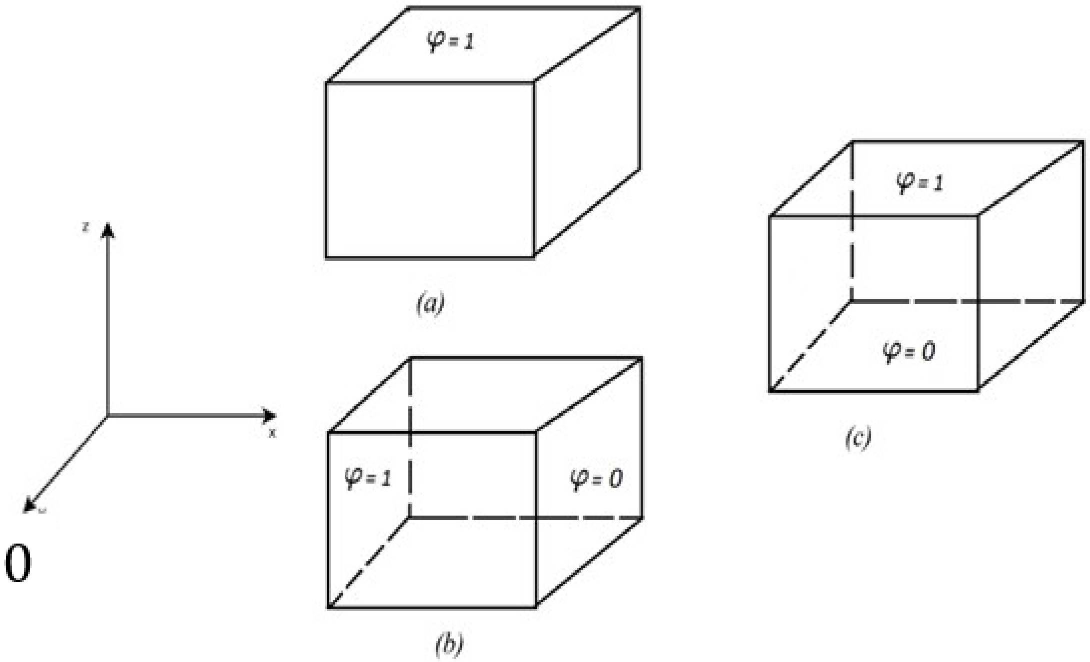
– ($\varphi(x, y, 1) = 1$

– **Case *b***

$$\varphi(0, y, z) = 1 \qquad \varphi(1, y, z) = 0$$

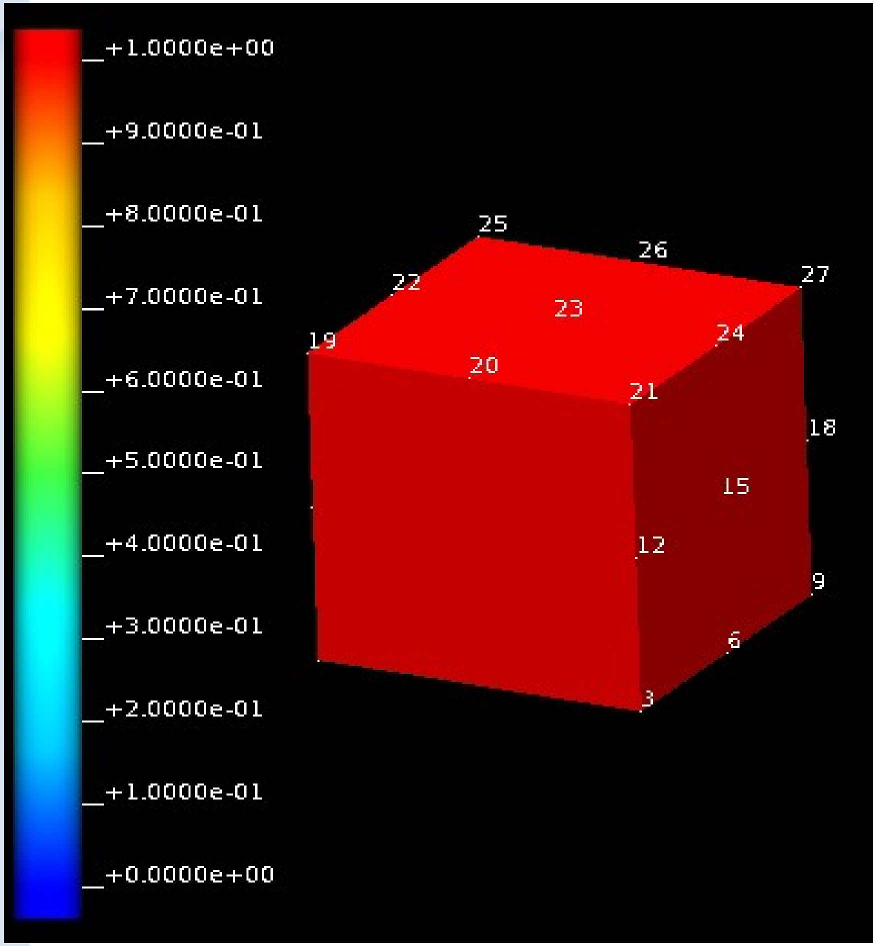
– **Case *c***

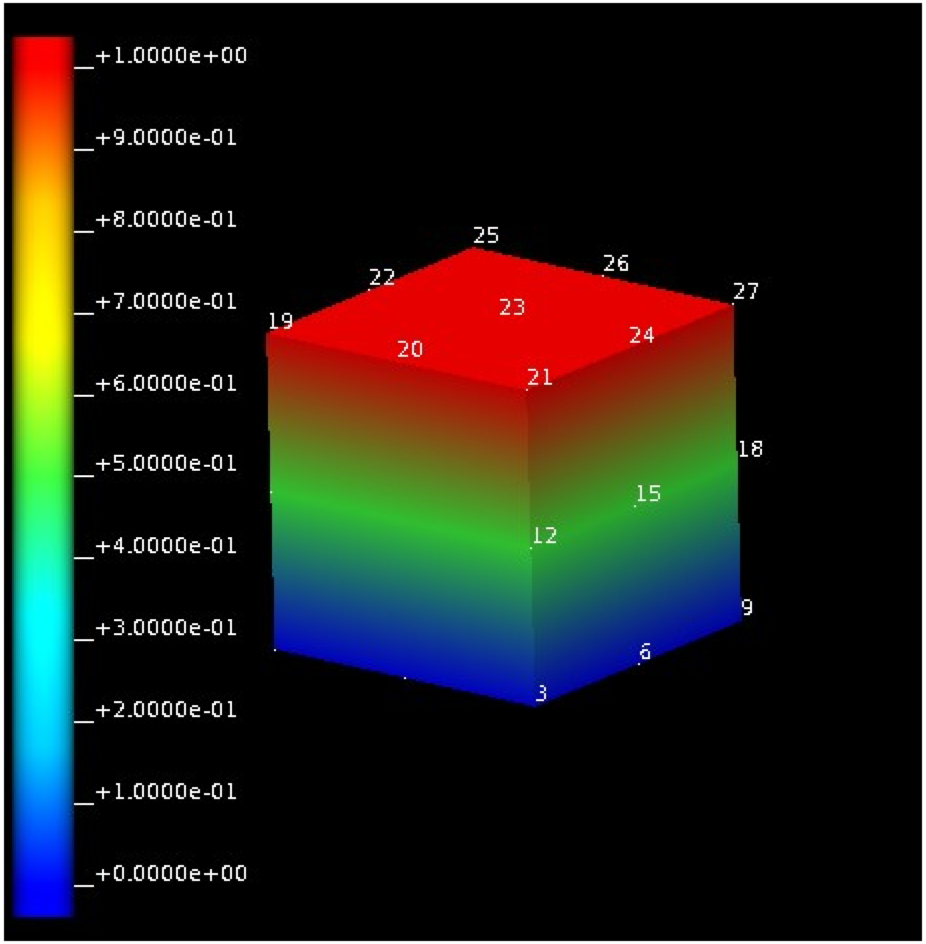
$$\varphi(x, y, 1) = 1 \qquad \varphi(x, y, 0) = 0$$



4.1 Laplace Problem (continued)

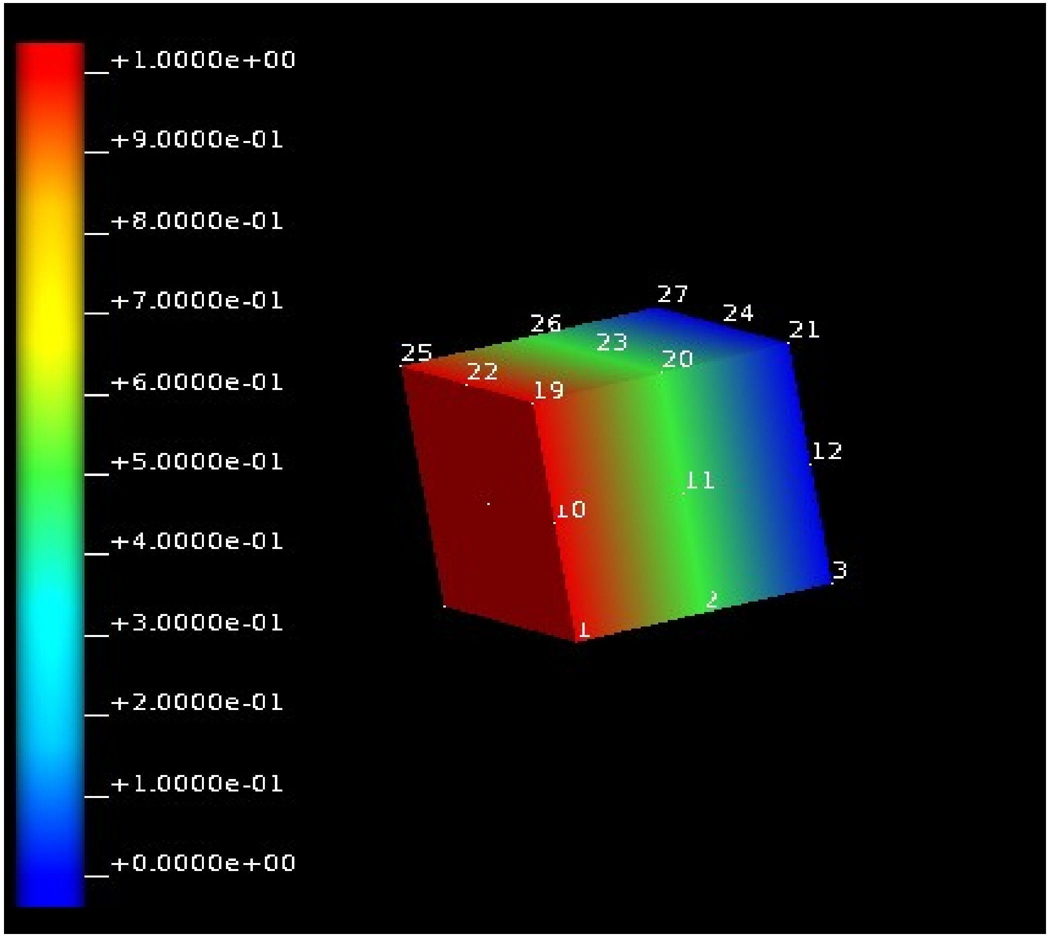
Case (a): $\varphi(x, y, 1) = 1$ Case (c): $\varphi(x, y, 1) = 1$ $\varphi(x, y, 0) = 0$





4.1 Laplace Problem (Continued)

Case (b): $\varphi(0, y, z) = 1 \quad \varphi(1, y, z) = 0$

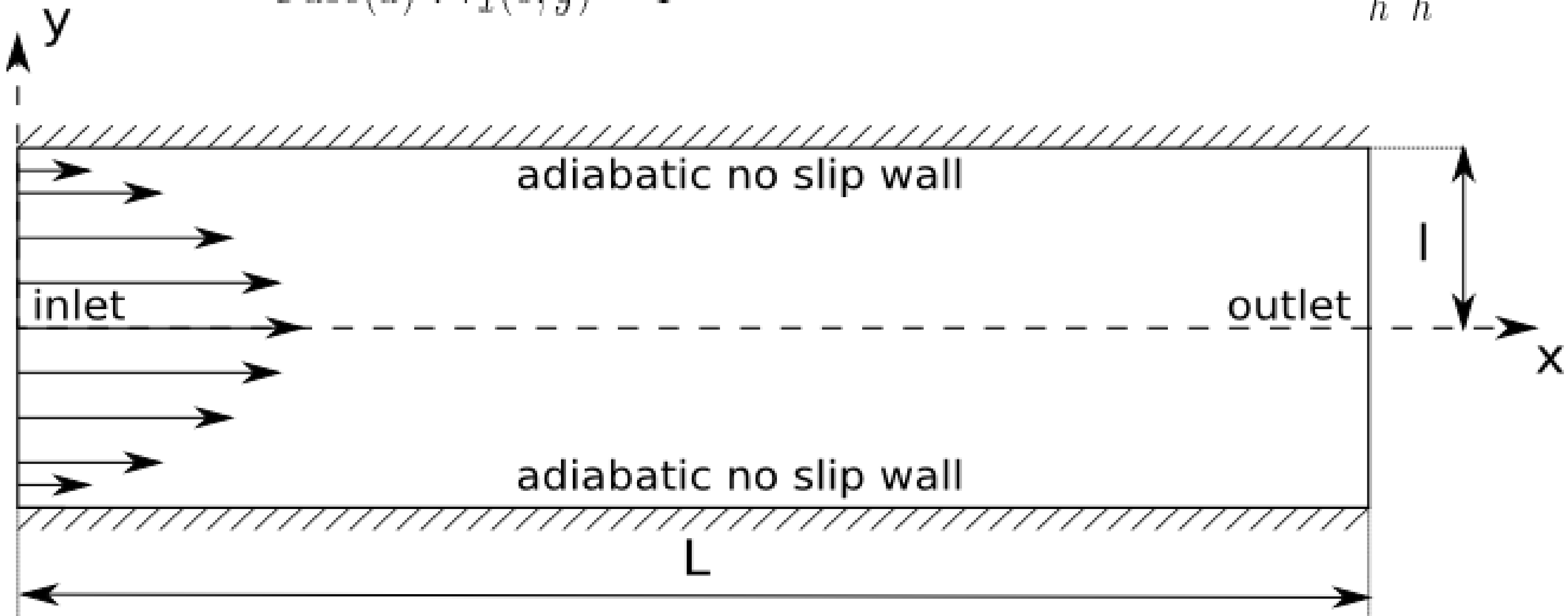


4.2 Case Study # 2
2D Fluid Flow Problem

- Governing equation to be solved

$$\underbrace{\rho\left(\frac{\partial \vec{u}}{\partial t} + \vec{u}\cdot\nabla\vec{u}\right)}_{\text{Acceleration}} = \underbrace{-\nabla p}_{\text{Pressure}} + \underbrace{\nu\,\triangle\,\vec{u}}_{\text{Viscosity}}$$

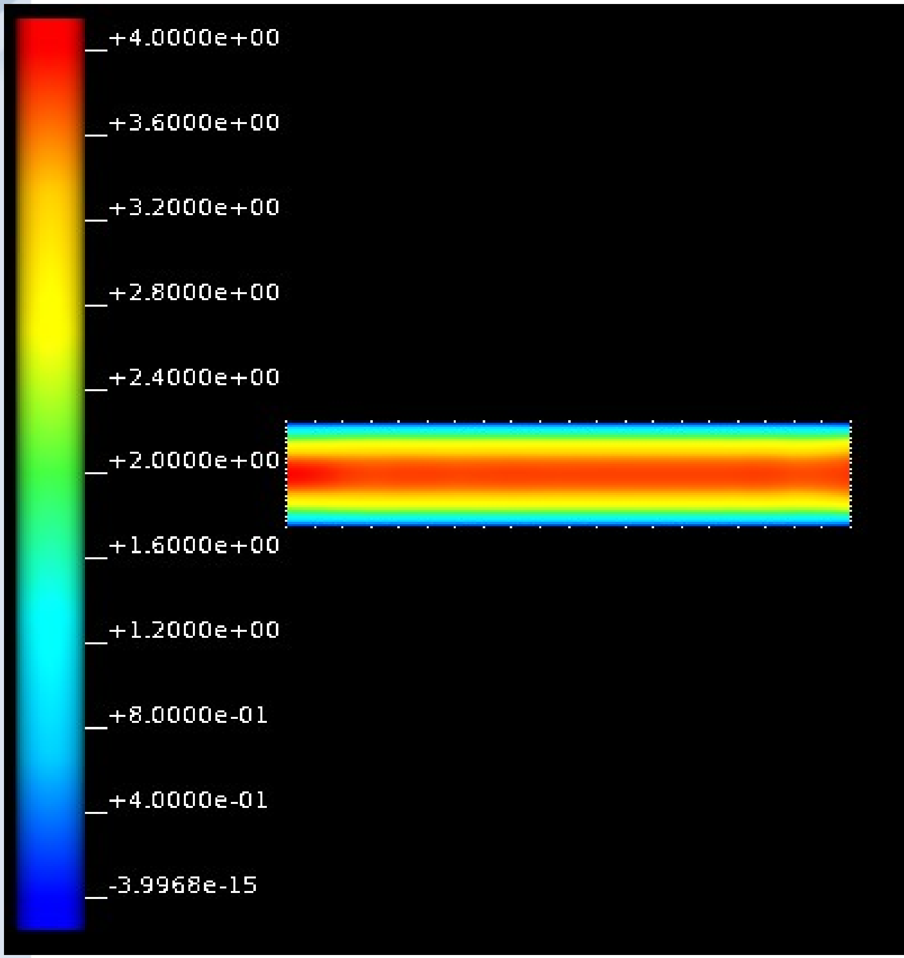
$$Case(a) : V_x(0,y) = 4 \qquad Case(b) : V_x(0,y) = 4V_{max}\frac{y}{h}\left(\frac{y}{h}-1\right)$$

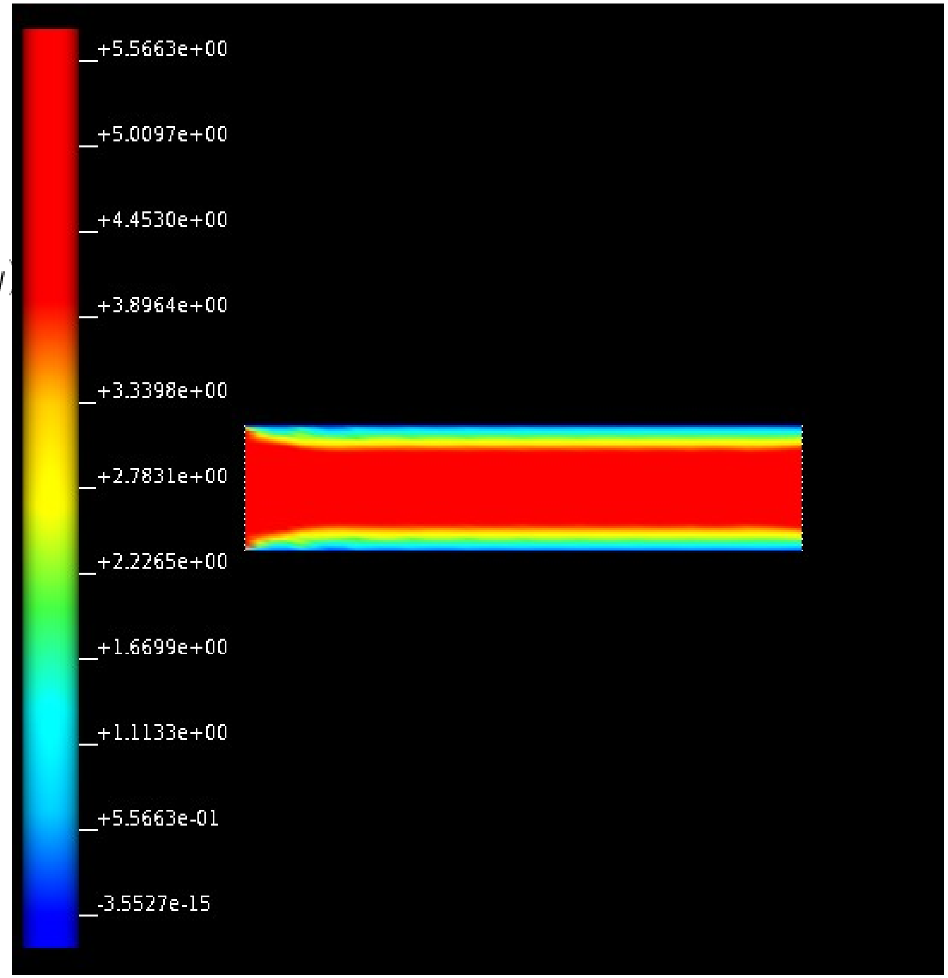


2d Fluid Flow Problem (Conti.)

$Case(b) : V_x(0, y) = 4V_{max}\frac{y}{h}(\frac{y}{h} - 1)$

$Case(a) : V_x(0, y) = 4$





CASE STUDY 3

- Will be ADDED later today

Conclusion and Future Research

- The developed parsing algorithm can ideally solve Fluid Mechanics , Solid Mechanics and Laplace problems.
- As of now, it cannot be guaranteed that the algorithm is bug free.
- Over the course of following months, the algorithm will be further developed and tested to resolve **potential bugs** and solve a **fluid solid interaction study**.