```python
import StandardLibrary as STDLIB
import StandardMaterialsLibrary as STDMTLLIB
import postProcess as POSTPRO
import csv
import numpy as np
from numpy.linalg import inv

# INPUTS
# ------------------------------------------------------------------------------------------
# The name of the input file
input_file = 'TrussProblem.sinp'

# Post processign info
deform_factor = 100000

# ------------------------------------------------------------------------------------------
# INPUTS END


name_ip_file = input_file[:-5]

# Read entire input file
all_lines = STDLIB.readFile(input_file)

# Parse the input file for 'keywords'
[keyword_lines, all_keywords,comment_lines, all_asterix] = STDLIB.parseKeywords(input_file)

# Read the Nodes
Nodes = STDLIB.readNodes(all_lines,all_keywords,keyword_lines,all_asterix)

# Read the elements
Elements = STDLIB.readElements(all_lines,all_keywords,keyword_lines,all_asterix)

# Read the linear elastic material
[E,nu] = STDLIB.read_elastic_material(all_lines,all_keywords,keyword_lines,all_asterix)


# Count the number of nodes
nnd = len(Nodes)
nel = len(Elements[0])

# Identify the element
name_of_Element = Elements[1]
currentElement = STDLIB.elementsLibrary(name_of_Element)

# Number of nodes per element
nne = currentElement.nne

# Number of degrees of freedom per node
nodof = currentElement.nodof

# Number of degrees of freedom per element
eldof = nne*nodof

'''
# Beam thickness in m
thick = 0.01

# Number of sampling points
num_gauss_points = 2;

# Form the elastic matrix for plane stress
dee = STDMTLLIB.formdsig(E,nu)

```

Make this section as an automatic input from the input file.

For the D-matrix according to element type.

```
# Create Node sets
# -----------------
NodeSets = STDLIB.createNodeSets(all_lines,all_keywords,keyword_lines,all_asterix)


# --------------------
# Boundary conditions
# --------------------

# Read the boundary conditions
BCS_NodeSet = STDLIB.read_BCS(all_lines,all_keywords,keyword_lines,all_asterix)

# Apply the Boundary Conditions
nf = STDLIB.apply_BCS(nnd,nodof,Nodes,NodeSets,BCS_NodeSet)

# Count the free degrees of freedom (Size of the stiffness matrix)
active_dof = 0

for i in range(0,nnd):
    for j in range(0,nodof):
        if nf[i,j] != 0:
            active_dof=active_dof+1
            nf[i,j]=active_dof

# -------
# Loading
# -------

# Read the node sets where the concentrated load sets are applied
[Cload_NodeSet_list, Cload_dof_mag]= STDLIB.read_Cloads(all_lines,all_keywords,keyword_lines,all_asterix)

# Apply the actual loading
Nodal_loads = STDLIB.apply_cloads(nnd,nodof,Nodes,NodeSets,Cload_NodeSet_list,Cload_dof_mag)


# -------------------------------------------------------------------------------
# Assemble the global force vector
# This force vector will have one column and active_dof-rows

force_global = np.zeros(shape=(active_dof,1))

for i in range(0,nnd):

    if nf[i][0] != 0:
        force_global[int(nf[i][0])-1] = Nodal_loads[i][0]

    if nf[i][1] != 0:
        force_global[int(nf[i][1])-1] = Nodal_loads[i][1]

# ------------------------------------------------------------------------------
# Assembly of the global stiffness matrix
# ------------------------------------------------------------------------------

# Collect the sampling points
samp = STDLIB.gaussPoints(num_gauss_points)

# Initialize the global stiffness matrix
KK = np.zeros(shape=(active_dof,active_dof))

# Form the element stiffness matrix and then assemble the global stiffness matrix
for i in range(0,nel):
    # Extract the coordinates of the element and the steering vector
    [coords,g] = STDLIB.elem_Q4(i,Nodes,Elements,nne,nodof,nf)

    # Initialize the element stiffness matrix
```

```
    ke = np.zeros(shape=(eldof,eldof))

    # Calculate the element stiffness matrix at each Gauss point
    for ig in range(0,num_gauss_points):
        for jg in range(0,num_gauss_points):

            [der_xi_eta, shapeFun] = STDLIB.fmQ4_lin(samp,ig,jg)

            # For the jacobian matrix
            jac = der_xi_eta.dot(coords)

            # Compute the inverse of the Jacobian matrix
            jac_inv = inv(jac)
```

Change this for a Truss Element

```
            # Compute the derivatives of the shape functions
            der_x_y = jac_inv.dot(der_xi_eta)

            # Form the B-matrix
            bee = STDLIB.formbee_Q4_lin(der_x_y,nne,eldof)

            # Integrate the stiffness matrix
            wi = samp[ig][1]
            wj = samp[ig][1]
            d = np.linalg.det(jac)

            ke = np.add(ke, reduce(np.dot, [d, thick, wi, wj, bee.transpose(), dee,bee]))

    # Form the global stiffness matrix
    KK = STDLIB.form_KK(KK,ke,g,eldof)

# Invert the global stiffness matrix and find the unknown displacements
delta = inv(KK).dot(force_global)

# Seperate the displacements into its componenets
# ------------------------------------------------
node_disp = STDLIB.seprarate_disp(nodof,nnd,delta,nf)

nodesFinal = Nodes[...,1:] + deform_factor*node_disp


# Name of the output database
name_output_db = name_ip_file + '.msh'
POSTPRO.write_gmsh_file(name_output_db,nnd,Nodes,nodesFinal,node_disp,nel,Elements)
'''
```

The output processor currently only calculates displacement. Add Stress also an an output.