Pipeline Implementation Part - I

First stage implementation

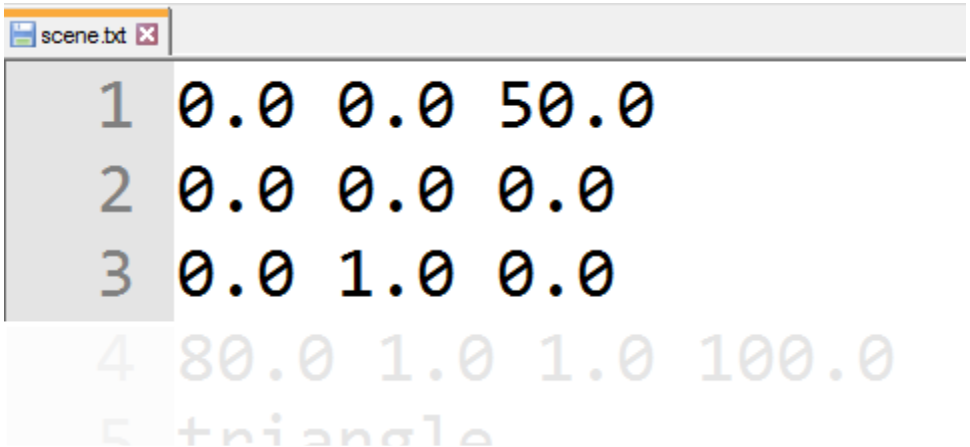
Acknowledgement: Lec Ishat-E-Rabban

Pipeline Implementation Part - I

- You will be given a scene.txt file as input
- It will contain triangle points as well as transformation commands
- You will transform the points based on transformation matrix by writing a C++ program
- Write the transformed points in stage1.txt file (See sample test case)

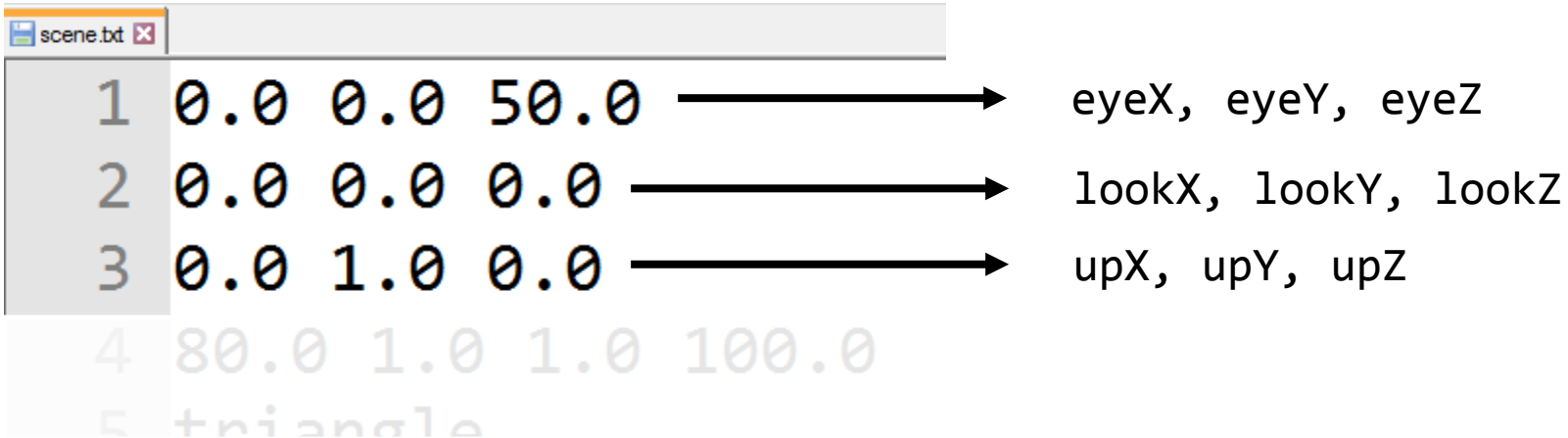
Reading `scene.txt`

Line 1 to 3: parameters of the `gluLookAt` function



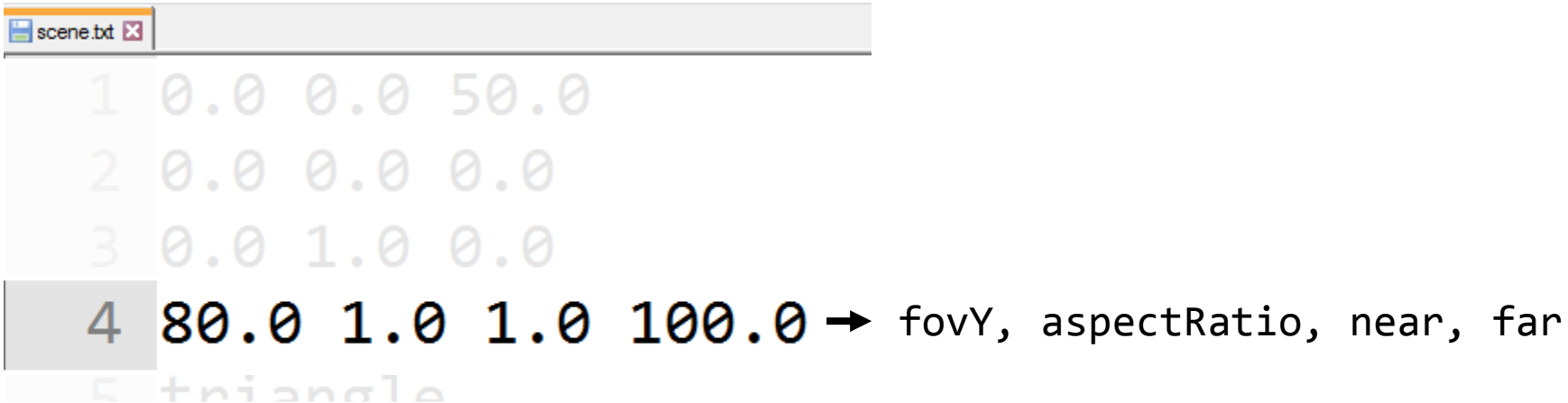
Reading `scene.txt`

Line 1 to 3: parameters of the `gluLookAt` function



Reading `scene.txt`

Line 4: parameters of the `gluPerspective` function



Reading `scene.txt`

Other commands : (1/7) Triangle

```
triangle
0.0 0.0 0.0
5.0 0.0 0.0
0.0 5.0 0.0
```

Equivalent to the following OpenGL code:

```
glBegin(GL_TRIANGLE);{
        glVertex3f(p1.x, p1.y, p1.z);
        glVertex3f(p2.x, p2.y, p2.z);
        glVertex3f(p3.x, p3.y, p3.z);
}glEnd();
```

Reading `scene.txt`

Other commands : (2/7) Translate

```
translate
10.0 0.0 0.0
```

Equivalent to the following OpenGL code:

```
glTranslatef(tx, ty, tz)
```

Reading scene.txt

Other commands : (3/7) Scale

```
scale
2.0 2.0 2.0
```

Equivalent to the following OpenGL code:

```
glScalef(sx, sy, sz)
```

Reading `scene.txt`

Other commands : (4/7) Rotate

```
rotate
60.0 -2.0 3.0 -4.0
```

Equivalent to the following OpenGL code:

```
glRotatef(angle, ax, ay, az)
```

Reading `scene.txt`

Other commands : (5, 6, 7) push, pop, end

```
0.0 5.0 0.0
pop
push
rotate
60.0 -2.0 3.0 -4.0
triangle
0.0 0.0 0.0
5.0 0.0 0.0
0.0 5.0 0.0
pop
end
```

push → Equivalent to **glPushMatrix** OpenGL command

pop → Equivalent to **glPopMatrix** OpenGL command

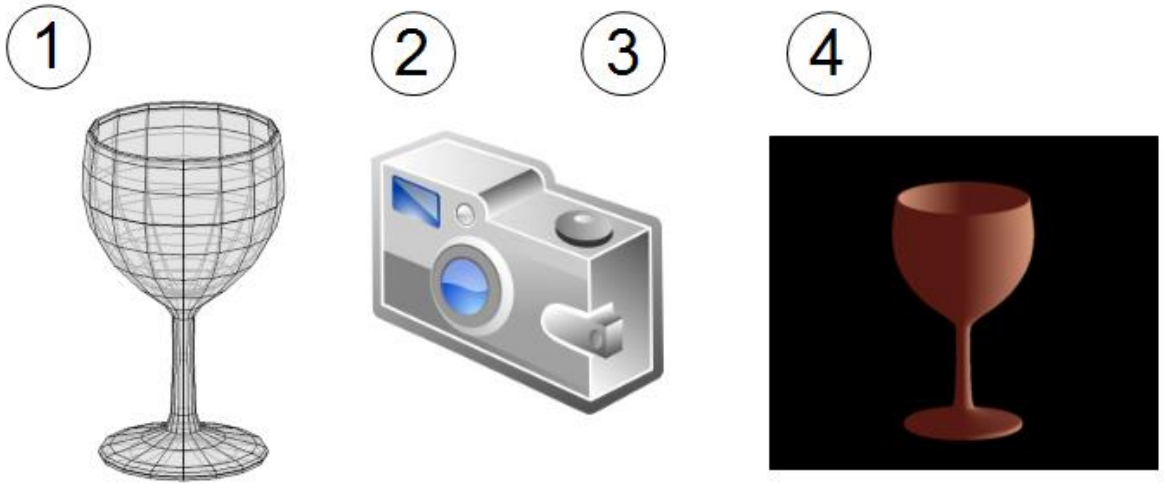
end → End of code

So, what’s the task?

The transformation stages: Camera analogy

So, what's the task?

The transformation stages: Camera analogy



(1) Positioning the model → Modeling transformation

(2) Positioning the camera → Viewing transformation

(3) Adjusting the zoom → Projection transformation

(4) Cropping the image → Viewport transformation

https://en.wikibooks.org/wiki/Cg_Programming/Vertex_Transformations

Stage 1: Modeling transformation

We'll start with an empty stack of matrix (4x4)

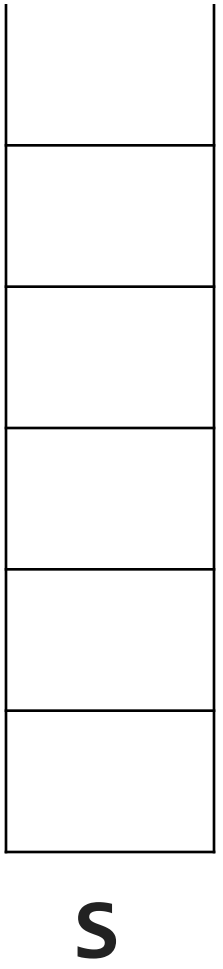
(Create a struct for your ease)

Stage 1: Modeling transformation

We'll start with an empty stack of matrix (4x4)

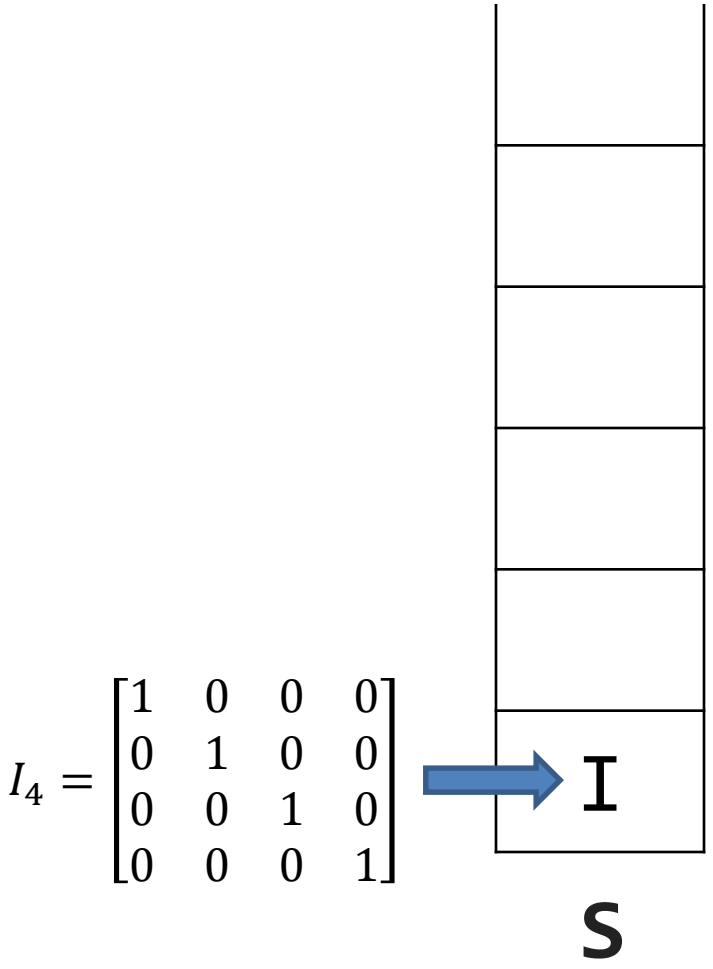
(Create a struct for your ease)

```
#include <stack>

struct Matrix
{
    //...
}

int main()
{
    stack <Matrix> S;
    //...
```



Stage 1: Modeling transformation

At the beginning, we push I_4 identity matrix in stack S

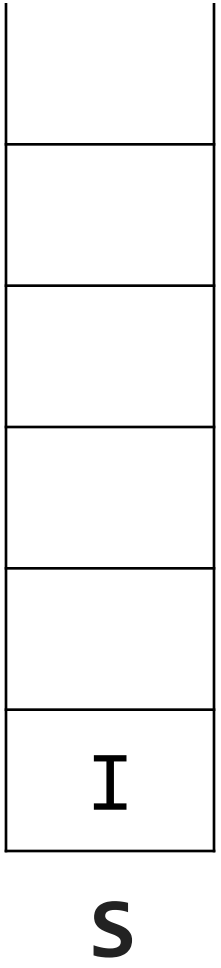


Stage 1: Modeling transformation

Parsing the scene.txt

```
while(1)
{
    input --> command;
    ...
    if (command == "scale")
      ➡ input scaling factors;
        generate scaling matrix T;
        S.push(product(S.top, T));
    ...

}
```

```
scale
2.0 2.0 2.0
```



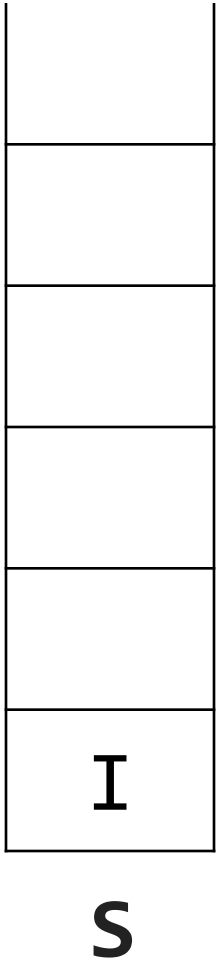
Stage 1: Modeling transformation

Parsing the scene.txt

```
while(1)
{
    input --> command;
    ...
    if (command == "scale")
        input scaling factors;
      ➡ generate scaling matrix T;
        S.push(product(S.top, T));
    ...

}
```

```
scale
sx sy sz
```

$$T = \begin{vmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

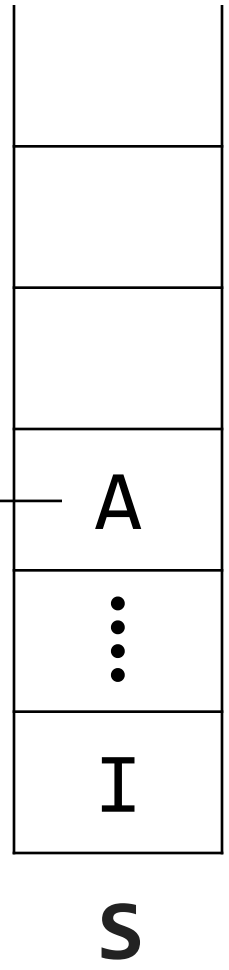


Stage 1: Modeling transformation

Parsing the scene.txt

```
while(1)
{
    input --> command;
    ...
    if (command == "scale")
        input scaling factors;
        generate scaling matrix T;
      ➡ S.push(product(S.top, T));
    ...

}
```

Temp matrix, B = A × T



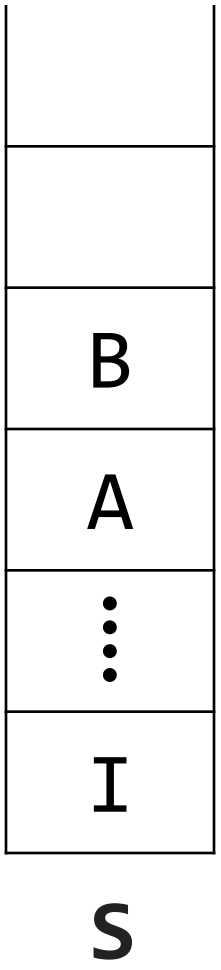
Stage 1: Modeling transformation

Parsing the scene.txt

```
while(1)
{
    input --> command;
    ...
    if (command == "scale")
        input scaling factors;
        generate scaling matrix T;
      ➡ S.push(product(S.top, T));
    ...

}
```

Temp matrix, B = A × T

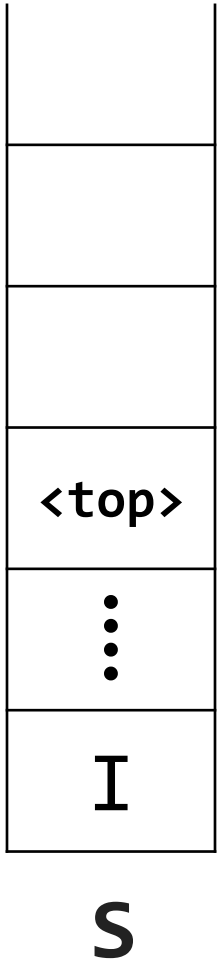
S.push(B)



Stage 1: Modeling transformation

Parsing the scene.txt

```
while(1)
{
    input --> command;
    ...
    if (command == "translate")
        input translate factors;
        generate translation matrix T;
        S.push(product(S.top, T));
    ...
}
```



*Corresponding matrix given on document

Stage 1: Modeling transformation

Parsing the scene.txt

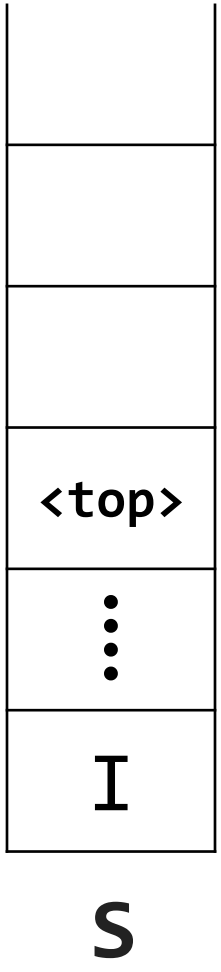
```
if (command == "rotate")
    input rotation factors;
    generate rotion matrix T;
    S.push(product(S.top, T));
```

*Corresponding matrix given on document

[You must normalize the rotational axis vector]

$$|v| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

Normalized vector, $\tilde{v} = \frac{v}{|v|}$



Stage 1: Modeling transformation

Parsing the scene.txt

```
if (command == "triangle")
     input three points;
     for each point P:
          P' <- transformPoint(S.top, P);
          output P' in file;
```

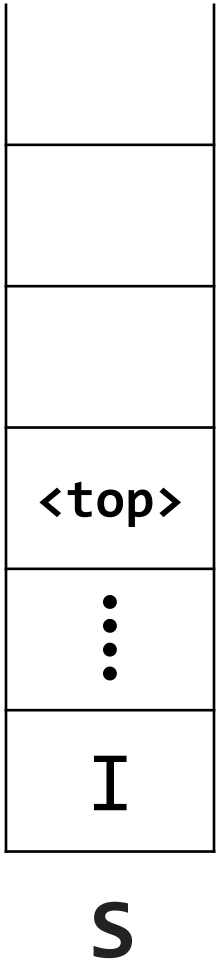
What does transformPoint() do?

It simply multiplies S.top matrix with P vector

(4x4) (4x1)

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax+by+cz+dw \\ ex+fy+gz+hw \\ ix+jy+kz+lw \\ mx+ny+oz+pw \end{bmatrix}$$

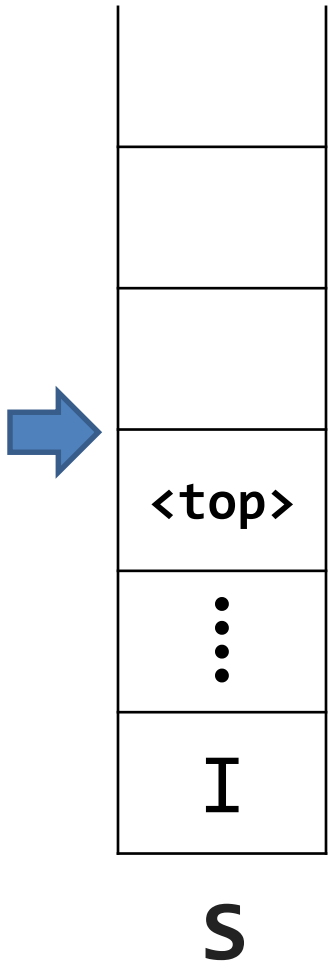
S.top P P’



Stage 1: Modeling transformation

Implementing push and pop

```
pop
push
rotate
60.0 -2.0 3.0 -4.0
triangle
```

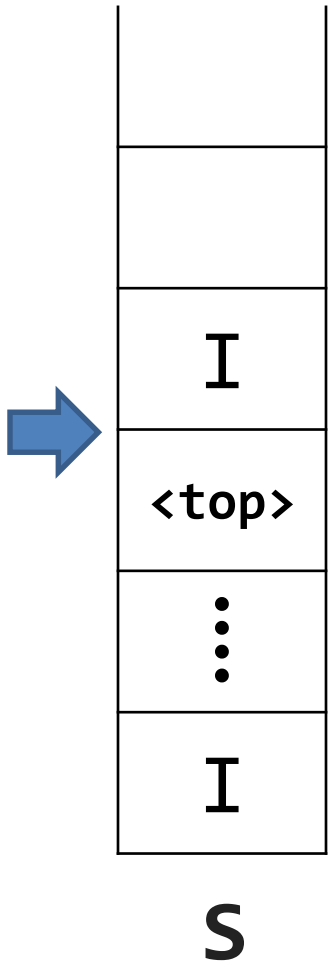


Keep a marker on where the push is called in stack,

Stage 1: Modeling transformation

Implementing push and pop

```
pop
push
rotate
60.0 -2.0 3.0 -4.0
triangle
```



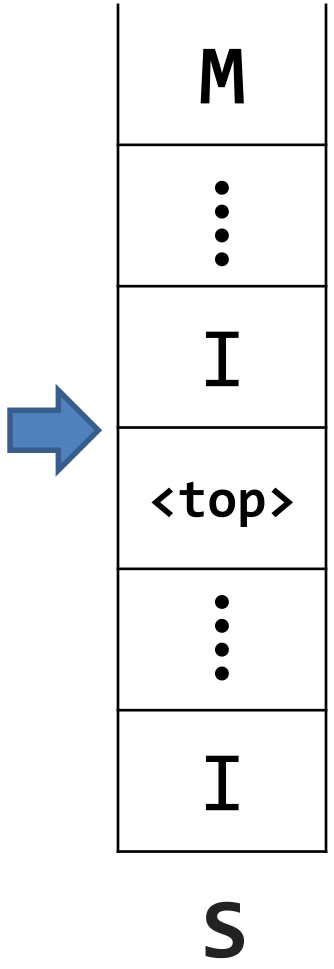
Keep a marker on where the push is called in stack,

and then push Identity matrix in S

Stage 1: Modeling transformation

Implementing push and pop

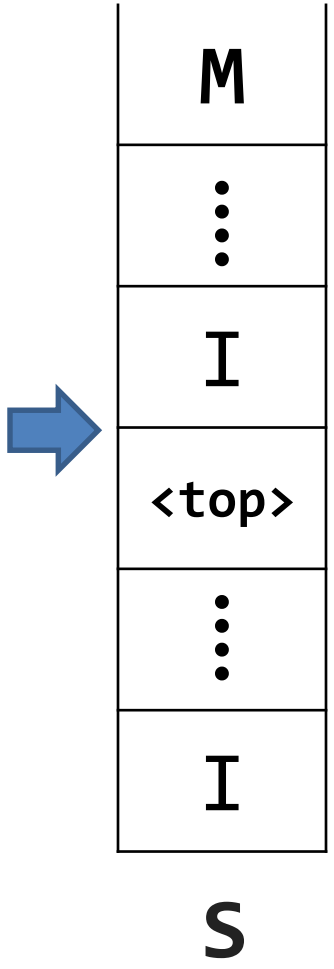
```
0.0 3.0 0.0
pop
push
rotate
60.0 -2.0 3.0 -4.0
triangle
```



Stage 1: Modeling transformation

Implementing push and pop

```
pop
push
rotate
60.0 -2.0 3.0 -4.0
triangle
pop
```

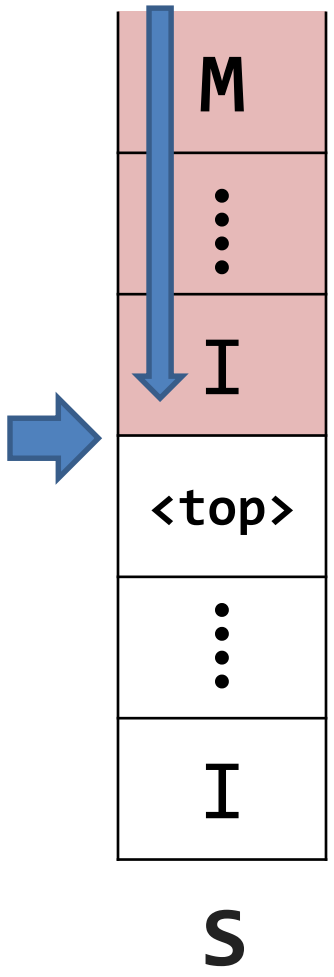


Stage 1: Modeling transformation

Implementing push and pop

```
pop
push
rotate
60.0 -2.0 3.0 -4.0
triangle
pop
```

Pop all up to marker

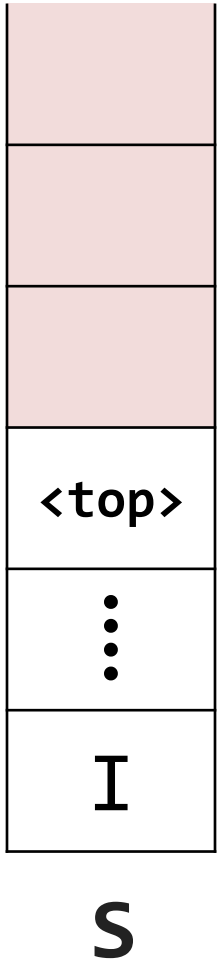


Stage 1: Modeling transformation

Implementing push and pop

```
pop
push
rotate
60.0 -2.0 3.0 -4.0
triangle
pop
```

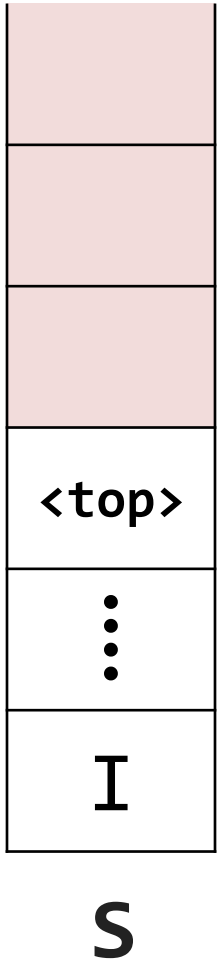
Pop all up to marker



Stage 1: Modeling transformation

Implementing push and pop

```
pop
push
rotate
60.0 -2.0 3.0 -4.0
triangle
pop
```



Your program should support multiple push/pop.

[Hint: Use multiple marker]

Instructions

1. Follow the provided document for actual implementation.

   Take help from this ppt if necessary
2. Don’t copy/try to copy

Submission time and date:

- You will only submit your cpp file
- The name of the cpp file will have the following format.

  `cse414[a/b]_task1_[your roll].cpp`

  Eg. For section A, roll 201514001, the file name will be:

  `cse414a_task1_201514001.cpp`

- Email the cpp file to: **submission.cse.mist@gmail.com** the day before the next class, **with the subject same as your filename (excluding .cpp)**

Marks Distribution

Submission	10%
File Operations and I/O Compatibility	20%
Translation	15%
Scaling	15%
Rotation	20%
Push and Pop	20%
	100%