NOTE: Your program *must* conform to everything presented in this document in order to receive any credit. You may, however, generate arbitrarily verbose debugging output of various kinds, including additional files. You may also wish to recognize additional command-line arguments for your own purpose. Just don't do anything that would prevent your program from being run by a shell script without human intervention. If you use additional files or command line arguments, you must modify `Makefile` so that your program compiles correctly. Remember that you will not receive credit if your program refuses to compile.

1 Description

In this assignment, you are to implement 3D transformation and viewing in graphics pipeline. Template programs along with `Makefile` are provided to help get you started. You may wish to obtain the following files

```
/home/cmpsc457/s19/hw4/template.cxx
/home/cmpsc457/s19/hw4/Makefile
```

on Sun Lab machines and modify them for this assignment. Use the following commands on your shell

```
make template
make run
make clean
```

to compile the program, to run the program, and to remove unnecessary files, respectively.

The objective of this homework is to give you an experience on implementing geometric transformations and 3D viewing. Hence, you are not allowed to use any of the following OpenGL functionalities:

- OpenGL matrix functions
- OpenGL transformations
- OpenGL 3D primitives
- OpenGL 3D projections

You are only allowed to use 2D OpenGL line primitive to draw objects. In fact, the only OpenGL calls that you should use for this homework are `glBegin(GL_LINE_LOOP)`, `glEnd()`, `glVertex2d()`, and `glutPostRedisplay()`. OpenGL and glut functions related to your user interface are allowed, but there should be **no** OpenGL 3D functionalities used in this homework. If you have any doubts or questions about what functions you are allowed to use, direct your questions to me, as the use of the wrong OpenGL functions will cost you significant marks. It is necessary to be very strict about this, since OpenGL 3D functionality makes this assignment trivial.

You are stronly recommended to follow the steps in the order presented below to complete this homework.

2 Projection (45 points)

You should start with the following functions used by `initCam()`:

- `SetViewMatrix()` : initializes $\mathbf{M}_{\text{cam}}$.
- `SetPerspMatrix()` : initializes $\mathbf{M}_{\text{p}}$.
- `SetOrthoMatrix()` : initializes $\mathbf{M}_{\text{o}}$.

After these functions are completed, you need to complete the function `drawFaces()`. In this function, you need to draw each face of the object, using `GL_LINE_LOOP`. Note that the coordinates of the vertices of the object are given in the object coordinate system. To obtain the pixel coordinates of the vertices, you first need to calculate the transformation matrix

$$\mathbf{M} = \mathbf{M}_{\text{o}} \cdot \mathbf{M}_{\text{p}} \cdot \mathbf{M}_{\text{cam}} \cdot \mathbf{M}_{\text{world}}$$

and transform each vertex with this matrix. Note that $\mathbf{M}_{\text{world}}$ is stored in `obj.frame` field. If you have completed these functions correctly, you should be able to see your object in the middle of the window.

You are only allowed to use 2D OpenGL line-loop primitive when you draw the faces of the object. In fact, the only OpenGL calls that you should use for drawing the given object on the screen are

`glBegin(GL_LINE_LOOP)`, `glEnd()`, `glVertex2d()`

In addition, you are not allowed to use any OpenGL viewing commands such as `glOrtho`, `glOrtho2D`, `gluLookAt`, `glFrustum`, `gluPerspective` etc, when completing `SetOrthoMatrix()`, `SetPerspMatrix`, and `SetViewMatrix`.

3 Transformation (45 points)

Complete the following functions in `template.cxx`

- `DeviceToWorld`
- `Translate_xy`
- `Translate_xz`
- `Scale`
- `Rotate`
- `SetRotMatrix`

You must not use any OpenGL transformation commands (such as, `glTranslate`, `glRotate`, `glScale`, etc) to transform your object. You need to implement transformations manually without using OpenGL transformation commands. Each function is described below.

It is important to understand that the original vertex coordinates of the object should never be changed by a sequence of these transformations. Instead, you record the sequence of transformations by multiplying the tranformation matrices to `obj.frame` in proper order. When you actually need to draw the object, you can apply `obj.frame` to vertices to get corresponding pixel coordinate of the vertices. If you change the original vertex coordinates in such a way that they cannot be recovered to the original values, `reset` functionality of the homework will not work properly.

3.1 `DeviceToWorld`

Converts device (screen or pixel) coordinates to world coordinates for user interaction.

3.2 `Translate_xy`

Translate the object in x–y plane in the same direction as the mouse moves while the left mouse button is held down.

3.3 `Translate_xz`

Translate the object in x–z plane in the same direction as the mouse moves while the shift key and the left mouse button are held down.

3.4 `Scale`

Scale the object in all dimensions uniformaly about its natural origin as the mouse moves while the right mouse button is held down. This means **scale all coordinates together by $1 + \alpha \cdot dx$, where α is a small number (e.g., 0.05) and dx is the displacement of the mouse in x direction.** Again, the fixed point of the scale should be the center of the object.

3.5 `Rotate`

Rotate the object as the mouse moves while middle mouse button is held down, using **Rolling Ball** mothod. Rolling Ball rotation is a rotation by an angle θ about an arbitrary axis $\hat{\mathbf{n}}$, such that

$$n_x = -\frac{dy}{dr}, \quad n_y = \frac{dx}{dr}, \quad n_z = 0$$

where, we define the mouse displacement dr as

$$dr = \sqrt{dx^2 + dy^2}\ .$$

The single free parameter of the algorithm is the effective rolling ball radius, R, which determines the sensitivity of the rotation angle to the displacement dr (Try between 1 and 10 in world coordinate units). We choose the rotation angle θ to be

$$\theta = \arctan \frac{dr}{R}$$

3.6 `SetRotMatrix`

Rotate a point by an angle θ about an arbirary axis $\hat{\mathbf{n}} = (n_x, n_y, n_z)$, where $\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = 1$, using the following matrix:

$$\begin{bmatrix} \cos\theta + n_x^2(1-\cos\theta) & n_x n_y(1-\cos\theta) - n_z\sin\theta & n_x n_z(1-\cos\theta) + n_y\sin\theta \\ n_y n_x(1-\cos\theta) + n_z\sin\theta & \cos\theta + n_y^2(1-\cos\theta) & n_y n_z(1-\cos\theta) - n_x\sin\theta \\ n_z n_x(1-\cos\theta) - n_y\sin\theta & n_z n_y(1-\cos\theta) + n_x\sin\theta & \cos\theta + n_z^2(1-\cos\theta) \end{bmatrix}$$

The key & mouse combinations are summarized in the following table:

Modifier	Button	Key	Operation
	left		Translate_xy
shift	left		Translate_xz
	right		Scale
	middle		Rotate
		r	Reset
		p	Toggle projection

What to Submit

- Electronic Submission (Due: 11:59:59 PM, March 20, 2019)
  - Your source codes
    - `template.cxx`, `matrices.cxx`, `matrices.h`
    - Additional source code files if you created any
    - Put your name and email address in every file you are submitting
  - `Makefile`
    - Put your name and email address in it.
    - If you implement extras, modify `Makefile` appropriately.
    - `Makefile` should compile your codes correctly
  - `Readme`
    - Put your name and email address in it.
    - Put the exact commands to compile and execute your program on separate lines so that a script can copy and paste to test your code automatically.
    - Put anything that you want me to know before grading your code, such as new features, extra credits, user interfaces, etc.
    - Plain ASCII text file (No doc files)
- Hardcopy Submissin (Due: At the beginning of the class on March 21, 2019)
  - Hardcopy of your `Readme`

Very important note: If your Makefile fails to compile and/or fails to run your code, you will not receive any credit for this homework. It is very important that your `Makefile` compiles your code without any errors and warnings, and run your program correctly. If you have additional source files or if you are accepting command-line arguments or anything, you must modify your `Makefile` appropriately so that it compiles/runs your code correctly.