General Instructions:

- Keep structure/class for things like triangle, sphere, point, vector, color etc.
- Keep all direction vectors as unit vector.
- You will need the following functions at least:
    - Function to calculate normal at a specific point of each type of object (triangle, sphere, square).
    - Function to calculate intersection of a ray (represented by a point and a direction vector) and each type of object.
    - Function to calculate reflected ray using a given incident ray and a reflection point on a object.

  I suggest taking help from slides of IER also to write the functions.

- This PDF is kind of a high(or mid) level overview to provide an idea. You can and may need to change things I suggested here depending on your approach.

Walkthrough:

- Take input and draw things in OpenGl (just like assignment 1). You won't get any mark for this but I believe it will help you to visualize things and to debug.
- Incorporate the camera operations from assignment 1. Ideally, there should be a camera position, up vector, look vector and right vector. Those should be updated appropriately with camera movement/rotation. The vectors should remain unit vectors and perpendicular to each other.
- When the '0' key will be pressed, you will need to generate a image. You should cast rays (straight lines) starting from camera position to all the directions to cover the whole screen. For these rays, the starting point is fixed (the camera position) but you will need to generate different direction vectors depending on which pixel you are drawing. If you are drawing the pixel at i^{th} row and j^{th} column, you may:
    - Generate the coordinates of that point and calculate the direction vector from camera position to that point.
    - Or, you can generate the direction vector by using scaled versions of your up, right, and look vector.
- Write a function, **F,** which will be a recursive function. F will take a ray as input (represented with starting point **P**, and direction vector **V**) and also it will keep track of current object (**currentObject**) it's working on. F will return a color (rgb component). The description of F is something like this:
    - Compute ambient, diffuse, and specular components at point P of currentObject according to IA's slides of our last class test.
    - Compute intersection points of **currentRay** (defiend by P and V) with each of the objects. Take the closest one, **intersectionPoint.** Let currentRay intersects with object **nextObject** at intersectionPoint. Find the **reflectedRay** of currentRay at intersectionPoint of nextObject. Call F with intesectionPoint, reflectedRay and nextObject. The returned value should be multiplied with the reflection component of

currentObject and added to its previously computed ambient, diffuse, and specular components. Then return this color.

There are corner cases. In some cases you won't have to calculate ambient, diffuse, specular components. In some cases there will be objects between point and light source; in those cases you won't get specular or diffuse component. Some rays may not intersect any object at all. Think about and handle all the cases carefully. Stop calculating the reflected ray and recursion call when maximum level of reflection (given in input) achieved. To get the color of a pixel, cast a ray from camera to that direction and call F with that ray.