1. Initialize SDL stuff (window, renderer, and texture)
2. Setup()
    a. Load obj file and its corresponding texture from png (using ung library)
    b. Set render mode, cull mode, fovy, fovx, aspect ratio, znear, zfar
    c. Make a perspective matrix
    d. Initialize camera frustum planes
    e. Add lighting
3. while(true) // game loop
    a. process_input()
        i. zoom in/zoom out, camera pitch and yaw, render mode, shading mode, quit
    b. update()
        i. deltaTime = timeSpent since last frame
        ii. Set rotate/translate/scale MESH in model space
        iii. get_camera_lookAt_target()
        iv. create cameraViewMatrix
        v. Build rotate/translate/scale model to world space matrix
        vi. for each face in MESH
            1. transform every vertex using modelToWorldMatrix
            2. transform every vertex w.r.t camera using camera view matrix
            3. perform culling if backface culling on
                a. dotProduct(cameraToVertex, normalToFace) < 0 → no render
            4. Flat shading → based on cameraRay and faceNormal determine color for face
            5. Gouraud Shading →based on cameraRay and vertex normal determine at each vertex
                a. Later use baycentric coordinates to interpolate pixels between vertex
            6. Create a polygon (3 sided) from the face
            7. Clip polygon against each of the 6 faces
                a. Take two vertex and one face, if one falls inside and other falls outside → find point of intersection and split into two line segments
                b. Keep track of interpolated tex coordinates as well
            8. Create triangles from the new polygon
            9. For every triangle do projection using projection matrix [Perspective Divide]
            10. Add triangle to drawCall List
        vii. Render()
            1. Clear z_buffer and color_buffer
            2. For all triangles in drawCall List
            3. Render every triangle using scanline algorithm
                a. Using baycentric coordinates and DDA line algorithm
                b. Draw every pixel with interpolated color and texture
    c. Call destructors for window and free the memory used for mesh data.