1. Problem statement
    - The first problem was to use a bezier patch to create a wireframe surface. This is meant to allow the user to move control points around which will affect the shape of the surface. By clicking any control point the user can use the keyboard to move that control point around which changes the shape.
    - The other problem was to use the opengl lighting engine to light up the bezier curve. The goal of this part was to use flat and smooth lighting to make the bezier surface light up and allow for the surface to be modified in real time as well as allowing for the light aspects to be changed.
    - The first extra credit was to allow the user to change any of the 16 control points
    - The second extra credit was to create an S using a bspline curve
    - The third extra credit was to read a texture and map it onto the bezier surface while also lighting up the texture.
2. Algorithm design
    - Bezier Patch
        i. Creates 16 control points
        ii. For each row calculate a bezier curve
        iii. Use each point calculated as a control point to calculate another bezier curve
        iv. Run that function for each point to calculate
        v. Render triangles from points calculated
    - Lighting
        i. Initialize opengl light with the given diffuse/specular/ambient/shininess
        ii. Set the material color to green with white for specular
        iii. Calculate normals for vertices by finding the average of every other triangle normal that uses this vertex
    - Change any control point
        i. Each control point has a unique color calculated based on its position
        ii. When a point is clicked the point is calculated based on the color
        iii. Whichever point is selected can be changed with the keyboard controls
    - B spline
        i. A function uses equations to calculate a point based on the u position in the b spline with 4 control points
        ii. for each 4 control points 0,1,2,3 1,2,3,4 etc. go from 0 to 1 at intervals and plot lines
        iii. Each control point has a unique color allowing it to be moved
        iv. NOTE: curve appears to go through all points only because first and last point aren't rendered.
    - Texture mapping
        i. The texture is read from the file
        ii. For each point in the bezier patch that points position is used to calculate what piece of the texture to use
        iii. the texture is mapped onto the surface

3. Important code
    - Bezier

```
Vec3f bezier_curve(std::array<Vec3f, 4> p, float u)
    Vec3f ret;
    float b1 = std::pow(1-u, 3);
    float b2 = 3*u*std::pow(1-u, 2);
    float b3 = 3*std::pow(u, 2)*(1-u);
    float b4 = std::pow(u, 3);
    ret.x += b1*p[0].x + b2*p[1].x + b3*p[2].x + b4*p[3].x;
    ret.y += b1*p[0].y + b2*p[1].y + b3*p[2].y + b4*p[3].y;
    ret.z += b1*p[0].z + b2*p[1].z + b3*p[2].z + b4*p[3].z;
    return ret;
```

This code creates 4 points based on the polynomials described in class and the float u which represents the parametric position on the curve. These points are used to calculate the x, y, and z of that parametric position. By running this for each row the bezier patch is created

- Lighting

```
void lighting():
    glClearColor (0.0, 0.0, 0.0, 0.0);
    if(render_type == 1 || render_type == 3)
        glShadeModel (GL_FLAT);
    else if(render_type == 2)
        glShadeModel (GL_SMOOTH);

    float pos[] = {-lightx, -lighty, -lightz, 0.0};
    glLightfv(GL_LIGHT0, GL_AMBIENT, LightAmbient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, LightDiffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, LightSpecular);
    glLightfv(GL_LIGHT0, GL_POSITION, pos);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
```

This code is for initializing the lighting. Mostly just uses built in functions to set up lighting. I had to make all light coordinates negative, I suspect this is because my normal values are negated.

```
Vec3f normal_avg(Vec3f p):
    float count = 0;
    float x = 0, y = 0, z = 0;
    for(const auto& t: triangles){
```

```
        if(t.contains(p)){
            x += t.normal.x;
            y += t.normal.y;
            z += t.normal.z;
            count++;
        }
    }
    return Vec3f(x/count, y/count, z/count)
```

This code is used for calculating the normal values of the triangles. The program needs to average the normals of all adjacent triangles together to find the best normal for each vertex so it loops through and checks to see if each point is in the triangle. if it is then it adds it to the average. This code is pretty slow because of how triangles are stored but could be sped up pretty easily.

4. Instructions
    - make
    - ./main
    - controls:
        i. click - select a point
        ii. w, s - change x coord
        iii. a, d - change z coord
        iv. r, f - change y coord
        v. z, x - move camera along z axis
        vi. t, g - move camera along y axis
        vii. k, i - move camera on z axis
        viii. j, l - move camera on x axis
        ix. menu
            1. change between lighting modes and wireframe
            2. change diffuse lighting
5. Images
    - Bezier surface
    - Flat lighting
    - Smooth lighting
    - Higher diffuse
    - Different angle lighting
    - Flower texture
    - Modified b spline S