Software engineering

Sahil Vaghasiya

202201083

GitHub link: https://github.com/processing/p5.js/blob/main/src/webgl/material.js

```
import p5 from '../core/main';
import * as constants from '../core/constants';
import './p5.Texture';
 * @method loadShader
 * @param {String} vertFilename path of the vertex shader to be
loaded.
 * @param {String} fragFilename path of the fragment shader to be
loaded.
 * @param {function} [successCallback] function to call once the
shader is loaded. Can be passed the
 *                                        <a
href="#/p5.Shader">p5.Shader</a> object.
 * @param {function} [failureCallback] function to call if the shader
fails to load. Can be passed an
 *                                        `Error` event object.
 * @return {p5.Shader} new shader created from the vertex and fragment
shader files.
 *
 * @example
 * <div modernizr='webgl'>
 * <code>
 * // Note: A "uniform" is a global variable within a shader program.
 *
 * let mandelbrot;
 *
 * // Load the shader and create a p5.Shader object.
 * function preload() {
 *   mandelbrot = loadShader('assets/shader.vert',
'assets/shader.frag');
 * }
 *
```

```
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   // Compile and apply the p5.Shader object.
 *   shader(mandelbrot);
 *
 *   // Set the shader uniform p to an array.
 *   mandelbrot.setUniform('p', [-0.74364388703, 0.13182590421]);
 *
 *   // Set the shader uniform r to the value 1.5.
 *   mandelbrot.setUniform('r', 1.5);
 *
 *   // Add a quad as a display surface for the shader.
 *   quad(-1, -1, 1, -1, 1, 1, -1, 1);
 *
 *   describe('A black fractal image on a magenta background.');
 * }
 * </code>
 * </div>
 *
 * <div>
 * <code>
 * // Note: A "uniform" is a global variable within a shader program.
 *
 * let mandelbrot;
 *
 * // Load the shader and create a p5.Shader object.
 * function preload() {
 *   mandelbrot = loadShader('assets/shader.vert',
'assets/shader.frag');
 * }
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   // Use the p5.Shader object.
 *   shader(mandelbrot);
 *
 *   // Set the shader uniform p to an array.
 *   mandelbrot.setUniform('p', [-0.74364388703, 0.13182590421]);
 *
 *   describe('A fractal image zooms in and out of focus.');
 * }
 *
 * function draw() {
```

```
 *   // Set the shader uniform r to a value that oscillates between 0
and 2.
 *   mandelbrot.setUniform('r', sin(frameCount * 0.01) + 1);
 *
 *   // Add a quad as a display surface for the shader.
 *   quad(-1, -1, 1, -1, 1, 1, -1, 1);
 * }
 * </code>
 * </div>
 */
p5.prototype.loadShader = function (
  vertFilename,
  fragFilename,
  successCallback,
  failureCallback
) {
  p5._validateParameters('loadShader', arguments);
  if (!failureCallback) {
    failureCallback = console.error;
  }

  const loadedShader = new p5.Shader();

  const self = this;
  let loadedFrag = false;
  let loadedVert = false;

  const onLoad = () => {
    self._decrementPreload();
    if (successCallback) {
      successCallback(loadedShader);
    }
  };

  this.loadStrings(
    vertFilename,
    result => {
      loadedShader._vertSrc = result.join('\n');
      loadedVert = true;
      if (loadedFrag) {
        onLoad();
      }
    },
    failureCallback
  );
```

```
  this.loadStrings(
    fragFilename,
    result => {
      loadedShader._fragSrc = result.join('\n');
      loadedFrag = true;
      if (loadedVert) {
        onLoad();
      }
    },
    failureCallback
  );

  return loadedShader;
};
```

1. **Data Referencing Errors**

   None found. The variables used in the function (vertFilename, fragFilename, successCallback, failureCallback) are referenced correctly and appropriately within the scope.

2. **Data Declaration Errors**

   None found. All required data structures and variables are declared properly. For example, loadedShader, loadedFrag, loadedVert, and onLoad are declared correctly before use.

3. **Computation Errors**

   None found. There are no mathematical computations or logical operations where computation errors could arise in this context.

4. **Comparison Errors**

   None found. There are no explicit comparisons in the code block that would result in comparison errors (like comparing different types).

5. **Control Flow Errors**

   None found. The control flow logic is clean, with conditions ensuring that both loadedFrag and loadedVert are true before proceeding with

onLoad(). The callback mechanisms work synchronously after both shaders are loaded.

6. **Interface Errors**

   Potential Issue: If the loadStrings method or the Shader constructor is not defined or has been changed, it could lead to issues. The method heavily depends on p5.loadStrings() and p5.Shader(), which must be implemented correctly. If not, it might result in errors or undefined behavior.

7. **Input/Output Errors**

   None found. The shader files are loaded through the loadStrings() function, which seems to be correctly implemented for handling input from files (vertex and fragment shader files). There's no indication of incorrect I/O operations in this snippet.

```
* @method createShader
 * @param {String} vertSrc source code for the vertex shader.
 * @param {String} fragSrc source code for the fragment shader.
 * @param {Object} [options] An optional object describing how this
shader can
 * be augmented with hooks. It can include:
 *   - `vertex`: An object describing the available vertex shader
hooks.
 *   - `fragment`: An object describing the available frament shader
hooks.
 * @returns {p5.Shader} new shader object created from the
 * vertex and fragment shaders.
 *
 * @example
 * <div modernizr='webgl'>
 * <code>
 * function setup() {
 *   createCanvas(50, 50, WEBGL);
 *
 *   // Create a shader with hooks
 *   myShader = createShader(vertSrc, fragSrc, {
```

```
 *     fragment: {
 *       'vec4 getColor': '(vec4 color) { return color; }'
 *     }
 *   });
 *
 *   // Make a version of the shader with a hook overridden
 *   modifiedShader = myShader.modify({
 *     'vec4 getColor': `(vec4 color) {
 *       return vec4(0., 0., 1., 1.);
 *     }`
 *   });
 * }
 *
 * function draw() {
 *   noStroke();
 *
 *   push();
 *   shader(myShader);
 *   translate(-width/3, 0);
 *   sphere(10);
 *   pop();
 *
 *   push();
 *   shader(modifiedShader);
 *   translate(width/3, 0);
 *   sphere(10);
 *   pop();
 * }
 * </code>
 * </div>
 */
p5.prototype.createShader = function (vertSrc, fragSrc, options) {
  p5._validateParameters('createShader', arguments);
  return new p5.Shader(this._renderer, vertSrc, fragSrc, options);
};
```

1. **Data Referencing Errors**

   None found. The variables (vertSrc, fragSrc, options) are referenced correctly. There is no evidence of variables being referenced before assignment.

2. **Data Declaration Errors**

None found. The input parameters (vertSrc, fragSrc, options) are declared properly, and the function returns a new p5.Shader object as expected.

3. **Computation Errors**

   None found. There are no computations in the code snippet that might lead to errors.

4. **Comparison Errors**

   None found. The code does not involve any comparisons, so no errors of this type are present.

5. **Control Flow Errors**

   None found. The control flow is straightforward and functions as intended. The parameters are validated using p5._validateParameters, and the shader object is created immediately after.

6. **Interface Errors**

   Potential Issue: The function relies on p5._validateParameters and p5.Shader. If these internal methods or the shader creation interface (p5.Shader) are not correctly implemented or malfunction, the function could fail to return a valid shader. However, this potential issue depends on the internal workings of the library and is beyond the scope of this specific function.

7. **Input/Output Errors**

   None found. The shader source strings (vertSrc and fragSrc) are passed as inputs, and the function correctly constructs the shader object without direct file I/O operations.

```
 * @method createFilterShader
 * @param {String} fragSrc source code for the fragment shader.
 * @returns {p5.Shader} new shader object created from the fragment
shader.
 *
 * @example
 * <div modernizr='webgl'>
 * <code>
 * function setup() {
 *   let fragSrc = `precision highp float;
 *   void main() {
 *     gl_FragColor = vec4(1.0, 1.0, 0.0, 1.0);
 *   }`;
 *
 *   createCanvas(100, 100, WEBGL);
 *   let s = createFilterShader(fragSrc);
 *   filter(s);
 *   describe('a yellow canvas');
 * }
 * </code>
 * </div>
 *
 * <div modernizr='webgl'>
 * <code>
 * let img, s;
 * function preload() {
 *   img = loadImage('assets/bricks.jpg');
 * }
 * function setup() {
 *   let fragSrc = `precision highp float;
 *
 *   // x,y coordinates, given from the vertex shader
 *   varying vec2 vTexCoord;
 *
 *   // the canvas contents, given from filter()
 *   uniform sampler2D tex0;
 *   // other useful information from the canvas
 *   uniform vec2 texelSize;
 *   uniform vec2 canvasSize;
 *   // a custom variable from this sketch
 *   uniform float darkness;
 *
 *   void main() {
 *     // get the color at current pixel
 *     vec4 color = texture2D(tex0, vTexCoord);
```

```
 *     // set the output color
 *     color.b = 1.0;
 *     color *= darkness;
 *     gl_FragColor = vec4(color.rgb, 1.0);
 *   }`;
 *
 *   createCanvas(100, 100, WEBGL);
 *   s = createFilterShader(fragSrc);
 * }
 * function draw() {
 *   image(img, -50, -50);
 *   s.setUniform('darkness', 0.5);
 *   filter(s);
 *   describe('a image of bricks tinted dark blue');
 * }
 * </code>
 * </div>
 */
p5.prototype.createFilterShader = function (fragSrc) {
  p5._validateParameters('createFilterShader', arguments);
  let defaultVertV1 = `
    uniform mat4 uModelViewMatrix;
    uniform mat4 uProjectionMatrix;

    attribute vec3 aPosition;
    // texcoords only come from p5 to vertex shader
    // so pass texcoords on to the fragment shader in a varying
variable
    attribute vec2 aTexCoord;
    varying vec2 vTexCoord;

    void main() {
      // transferring texcoords for the frag shader
      vTexCoord = aTexCoord;

      // copy position with a fourth coordinate for projection (1.0 is
normal)
      vec4 positionVec4 = vec4(aPosition, 1.0);

      // project to 3D space
      gl_Position = uProjectionMatrix * uModelViewMatrix *
positionVec4;
    }
  `;
  let defaultVertV2 = `#version 300 es
```

```
    uniform mat4 uModelViewMatrix;
    uniform mat4 uProjectionMatrix;

    in vec3 aPosition;
    in vec2 aTexCoord;
    out vec2 vTexCoord;

    void main() {
      // transferring texcoords for the frag shader
      vTexCoord = aTexCoord;

      // copy position with a fourth coordinate for projection (1.0 is
normal)
      vec4 positionVec4 = vec4(aPosition, 1.0);

      // project to 3D space
      gl_Position = uProjectionMatrix * uModelViewMatrix *
positionVec4;
    }
  `;
  let vertSrc = fragSrc.includes('#version 300 es') ? defaultVertV2 :
defaultVertV1;
  const shader = new p5.Shader(this._renderer, vertSrc, fragSrc);
  if (this._renderer.GL) {
    shader.ensureCompiledOnContext(this);
  } else {
    shader.ensureCompiledOnContext(this._renderer.getFilterGraphicsLay
er());
  }
  return shader;
};
```

1. **Data Referencing Errors**

   Possible Issue: The fragment shader fragSrc assumes that variables like vTexCoord, tex0, texelSize, and canvasSize will always be provided by the context or by other parts of the WebGL pipeline. If any of these uniforms are not set or properly linked during execution, it could cause referencing errors or missing data in the shader.

2. **Data Declaration Errors**

Possible Issue: While the function itself declares the vertex shader string variables defaultVertV1 and defaultVertV2, it does not explicitly check whether the source code for the fragment shader fragSrc is valid or not. An invalid or empty fragSrc would still be passed to the p5.Shader constructor, which may fail silently or cause runtime errors when applied.

3. **Computation Errors**

   None Found.

4. **Comparison Errors**

   None found. The string comparison fragSrc.includes('#version 300 es') is handled correctly to select the appropriate vertex shader version.

5. **Control Flow Errors**

   None found. The control flow is clear, with the function validating the parameters, determining the vertex shader version, creating the shader, and ensuring the shader is compiled on the correct context. There are no errors in the flow of logic.

6. **Interface Errors**

   Potential Issue: The code relies on internal methods like p5._validateParameters, p5.Shader, and shader.ensureCompiledOnContext(). If these methods or the shader compilation process is faulty, it could lead to unexpected behavior, but this would depend on the internal implementation of the library rather than this function itself.

7. **Input/Output Errors**

   None found. The function processes strings and interacts with the WebGL renderer to compile the shader. There are no file I/O operations that could cause errors.

```
 *
 * let redGreen;
 * let orangeBlue;
 * let showRedGreen = false;
 *
 * // Load the shader and create two separate p5.Shader objects.
 * function preload() {
 *   redGreen = loadShader('assets/shader.vert', 'assets/shader-
gradient.frag');
 *   orangeBlue = loadShader('assets/shader.vert', 'assets/shader-
gradient.frag');
 * }
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   // Initialize the redGreen shader.
 *   shader(redGreen);
 *
 *   // Set the redGreen shader's center and background color.
 *   redGreen.setUniform('colorCenter', [1.0, 0.0, 0.0]);
 *   redGreen.setUniform('colorBackground', [0.0, 1.0, 0.0]);
 *
 *   // Initialize the orangeBlue shader.
 *   shader(orangeBlue);
 *
 *   // Set the orangeBlue shader's center and background color.
 *   orangeBlue.setUniform('colorCenter', [1.0, 0.5, 0.0]);
 *   orangeBlue.setUniform('colorBackground', [0.226, 0.0, 0.615]);
 *
 *   describe(
 *     'The scene toggles between two circular gradients when the user
double-clicks. An orange and blue gradient vertically, and red and
green gradient moves horizontally.'
 *   );
 * }
 *
 * function draw() {
 *   // Update the offset values for each shader.
 *   // Move orangeBlue vertically.
 *   // Move redGreen horizontally.
 *   orangeBlue.setUniform('offset', [0, sin(frameCount * 0.01) + 1]);
 *   redGreen.setUniform('offset', [sin(frameCount * 0.01), 1]);
 *
 *   if (showRedGreen === true) {
```

```
 *     shader(redGreen);
 *   } else {
 *     shader(orangeBlue);
 *   }
 *
 *   // Style the drawing surface.
 *   noStroke();
 *
 *   // Add a quad as a drawing surface.
 *   quad(-1, -1, 1, -1, 1, 1, -1, 1);
 * }
 *
 * // Toggle between shaders when the user double-clicks.
 * function doubleClicked() {
 *   showRedGreen = !showRedGreen;
 * }
 * </code>
 * </div>
 */
p5.prototype.shader = function (s) {
  this._assert3d('shader');
  p5._validateParameters('shader', arguments);

  s.ensureCompiledOnContext(this);

  if (s.isStrokeShader()) {
    this._renderer.userStrokeShader = s;
  } else {
    this._renderer.userFillShader = s;
    this._renderer._useNormalMaterial = false;
  }

  s.setDefaultUniforms();

  return this;
};
```

1. **Data Referencing Errors**

   None found. The input parameter s (the shader object) is referenced correctly throughout the function. It is ensured that the shader object exists and is valid.

2. **Data Declaration Errors**

None found. The function parameters and internal variables are declared appropriately. There is no issue with variable scope or declaration.

3. **Computation Errors**

   None found. The function does not contain complex computations that could lead to errors. The operations performed are primarily method calls and state changes.

4. **Comparison Errors**

   None found. There are no comparison operations in the function, so there are no associated errors.

5. **Control Flow Errors**

   None found. The control flow is straightforward and logically structured. It checks whether the shader is a stroke shader or a fill shader and assigns it to the appropriate renderer property.

6. **Interface Errors**

   Potential Issue: The function assumes that the ensureCompiledOnContext, isStrokeShader, and setDefaultUniforms methods on the shader object s are implemented correctly. If any of these methods are missing or fail, it could lead to unintended behavior.

7. **Input/Output Errors**

   None found. The function properly handles input in the form of the shader object and does not perform any direct I/O operations. It correctly sets the shader as either a stroke or fill shader based on its type.

```
* function setup() {
*   createCanvas(100, 100, WEBGL);
*
*   // Hide the video.
*   vid.hide();
*
*   // Set the video to loop.
*   vid.loop();
*
*   describe('A rectangle with video as texture');
* }
*
* function draw() {
*   background(0);
*
*   // Rotate around the y-axis.
*   rotateY(frameCount * 0.01);
*
*   // Apply the video as a texture.
*   texture(vid);
*
*   // Draw the rectangle.
*   rect(-40, -40, 80, 80);
* }
* </code>
* </div>
*
* <div>
* <code>
* let vid;
*
* // Load a video and create a p5.MediaElement object.
* function preload() {
*   vid = createVideo('assets/fingers.mov');
* }
*
* function setup() {
*   createCanvas(100, 100, WEBGL);
*
*   // Hide the video.
*   vid.hide();
*
*   // Set the video to loop.
*   vid.loop();
*
```

```
 *   describe('A rectangle with video as texture');
 * }
 *
 * function draw() {
 *   background(0);
 *
 *   // Rotate around the y-axis.
 *   rotateY(frameCount * 0.01);
 *
 *   // Set the texture mode.
 *   textureMode(NORMAL);
 *
 *   // Apply the video as a texture.
 *   texture(vid);
 *
 *   // Draw a custom shape using uv coordinates.
 *   beginShape();
 *   vertex(-40, -40, 0, 0);
 *   vertex(40, -40, 1, 0);
 *   vertex(40, 40, 1, 1);
 *   vertex(-40, 40, 0, 1);
 *   endShape();
 * }
 * </code>
 * </div>
 */
p5.prototype.texture = function (tex) {
  this._assert3d('texture');
  p5._validateParameters('texture', arguments);
  if (tex.gifProperties) {
    tex._animateGif(this);
  }

  this._renderer.drawMode = constants.TEXTURE;
  this._renderer._useNormalMaterial = false;
  this._renderer._tex = tex;
  this._renderer._setProperty('_doFill', true);

  return this;
};
```

1. **Data Referencing Errors**

   None found. The input parameter tex (the texture object) is referenced correctly within the function, ensuring it is valid before proceeding with operations.

2. **Data Declaration Errors**

   None found. The function parameters and internal references are declared appropriately. There is no ambiguity regarding variable scope.

3. **Computation Errors**

   None found. The function does not contain complex computations, and all operations performed are method calls or property settings.

4. **Comparison Errors**

   None found. There are no comparison operations within the function, so this category does not apply.

5. **Control Flow Errors**

   None found. The control flow is straightforward and properly manages the state transitions for setting the texture. The logic is clear and correctly structured.

6. **Interface Errors**

   Potential Issue: The function assumes that the tex object is compatible with the texture operations expected by the renderer. If tex does not support the required properties or methods, it may lead to unexpected behavior.

7. **Input/Output Errors**

   None found. The function handles input as expected and does not perform any direct I/O operations. It correctly sets the rendering context and prepares the texture for drawing.

```
 * <code>
 * // Apply the image as a texture.
 * texture(img);
 *
 * // Draw the rectangle.
 * beginShape();
 *
 * // Top-left.
 * // u: 0, v: 0
 * vertex(0, 0, 0, 0, 0);
 *
 * // Top-right.
 * // u: 300, v: 0
 * vertex(30, 0, 0, 300, 0);
 *
 * // Bottom-right.
 * // u: 300, v: 500
 * vertex(30, 50, 0, 300, 500);
 *
 * // Bottom-left.
 * // u: 0, v: 500
 * vertex(0, 50, 0, 0, 500);
 *
 * endShape();
 * </code>
 *
 * `textureMode()` changes the coordinate system for uv coordinates.
 *
 * The parameter, `mode`, accepts two possible constants. If `NORMAL`
is
 * passed, as in `textureMode(NORMAL)`, then the texture's uv
coordinates can
 * be provided in the range 0 to 1 instead of the image's dimensions.
This can
 * be helpful for using the same code for multiple images of different
sizes.
 * For example, the code snippet above could be rewritten as follows:
 *
 * <code>
 * // Set the texture mode to use normalized coordinates.
 * textureMode(NORMAL);
 *
 * // Apply the image as a texture.
 * texture(img);
 *
```

```
 * // Draw the rectangle.
 * beginShape();
 *
 * // Top-left.
 * // u: 0, v: 0
 * vertex(0, 0, 0, 0, 0);
 *
 * // Top-right.
 * // u: 1, v: 0
 * vertex(30, 0, 0, 1, 0);
 *
 * // Bottom-right.
 * // u: 1, v: 1
 * vertex(30, 50, 0, 1, 1);
 *
 * // Bottom-left.
 * // u: 0, v: 1
 * vertex(0, 50, 0, 0, 1);
 *
 * endShape();
 * </code>
 *
 * By default, `mode` is `IMAGE`, which scales uv coordinates to the
 * dimensions of the image. Calling `textureMode(IMAGE)` applies the
default.
 *
 * Note: `textureMode()` can only be used in WebGL mode.
 *
 * @method  textureMode
 * @param {Constant} mode either IMAGE or NORMAL.
 *
 * @example
 * <div>
 * <code>
 * let img;
 *
 * // Load an image and create a p5.Image object.
 * function preload() {
 *   img = loadImage('assets/laDefense.jpg');
 * }
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe('An image of a ceiling against a black background.');
```

```
* }
*
* function draw() {
*   background(0);
*
*   // Apply the image as a texture.
*   texture(img);
*
*   // Draw the custom shape.
*   // Use the image's width and height as uv coordinates.
*   beginShape();
*   vertex(-30, -30, 0, 0);
*   vertex(30, -30, img.width, 0);
*   vertex(30, 30, img.width, img.height);
*   vertex(-30, 30, 0, img.height);
*   endShape();
* }
* </code>
* </div>
*
* <div>
* <code>
* let img;
*
* // Load an image and create a p5.Image object.
* function preload() {
*   img = loadImage('assets/laDefense.jpg');
* }
*
* function setup() {
*   createCanvas(100, 100, WEBGL);
*
*   describe('An image of a ceiling against a black background.');
* }
*
* function draw() {
*   background(0);
*
*   // Set the texture mode.
*   textureMode(NORMAL);
*
*   // Apply the image as a texture.
*   texture(img);
*
*   // Draw the custom shape.
```

```
 *   // Use normalized uv coordinates.
 *   beginShape();
 *   vertex(-30, -30, 0, 0);
 *   vertex(30, -30, 1, 0);
 *   vertex(30, 30, 1, 1);
 *   vertex(-30, 30, 0, 1);
 *   endShape();
 * }
 * </code>
 * </div>
 */
p5.prototype.textureMode = function (mode) {
  if (mode !== constants.IMAGE && mode !== constants.NORMAL) {
    console.warn(
      `You tried to set ${mode} textureMode only supports IMAGE &
NORMAL `
    );
  } else {
    this._renderer.textureMode = mode;
  }
};
```

1. **Data Referencing Errors**

   None found. The input parameter mode is referenced correctly within the function to determine the texture mode.

2. **Data Declaration Errors**

   None found. The function parameters and internal references are properly declared, ensuring there is no ambiguity.

3. **Computation Errors**

   None found. The function is straightforward and does not involve complex calculations. It simply checks the value of mode.

4. **Comparison Errors**

   None found. The function checks the value of mode against two constants, IMAGE and NORMAL, without any issues.

5. **Control Flow Errors**

   None found. The flow control correctly handles the scenario where an invalid mode is passed, issuing a warning without disrupting the program flow.

6. **Interface Errors**

   Potential Issue: The function relies on the existence of the constants object, which should define IMAGE and NORMAL. If these constants are not properly defined or available, it may lead to issues.

7. **Input/Output Errors**

   None found. The function does not perform direct I/O operations and handles input (the mode) as expected.

```
 * </code>
 * </div>
 *
 * <div>
 * <code>
 * let img;
 *
 * function preload() {
 *   img = loadImage('assets/rockies128.jpg');
 * }
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe(
 *     'Four identical images of a landscape arranged in a grid. The
images are reflected horizontally and vertically, creating a
kaleidoscope effect.'
 *   );
 * }
 *
 * function draw() {
```

```
 *   background(0);
 *
 *   // Set the texture mode.
 *   textureMode(NORMAL);
 *
 *   // Set the texture wrapping.
 *   textureWrap(MIRROR);
 *
 *   // Apply the image as a texture.
 *   texture(img);
 *
 *   // Style the shape.
 *   noStroke();
 *
 *   // Draw the shape.
 *   // Use uv coordinates > 1.
 *   beginShape();
 *   vertex(-30, -30, 0, 0, 0);
 *   vertex(30, -30, 0, 2, 0);
 *   vertex(30, 30, 0, 2, 2);
 *   vertex(-30, 30, 0, 0, 2);
 *   endShape();
 * }
 * </code>
 * </div>
 *
 * <div>
 * <code>
 * let img;
 *
 * function preload() {
 *   img = loadImage('assets/rockies128.jpg');
 * }
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe(
 *     'Four identical images of a landscape arranged in a grid. The
top row and bottom row are reflections of each other.'
 *   );
 * }
 *
 * function draw() {
 *   background(0);
```

```
 *
 *   // Set the texture mode.
 *   textureMode(NORMAL);
 *
 *   // Set the texture wrapping.
 *   textureWrap(REPEAT, MIRROR);
 *
 *   // Apply the image as a texture.
 *   texture(img);
 *
 *   // Style the shape.
 *   noStroke();
 *
 *   // Draw the shape.
 *   // Use uv coordinates > 1.
 *   beginShape();
 *   vertex(-30, -30, 0, 0, 0);
 *   vertex(30, -30, 0, 2, 0);
 *   vertex(30, 30, 0, 2, 2);
 *   vertex(-30, 30, 0, 0, 2);
 *   endShape();
 * }
 * </code>
 * </div>
 */
p5.prototype.textureWrap = function (wrapX, wrapY = wrapX) {
  this._renderer.textureWrapX = wrapX;
  this._renderer.textureWrapY = wrapY;

  for (const texture of this._renderer.textures.values()) {
    texture.setWrapMode(wrapX, wrapY);
  }
};

 * @method normalMaterial
 * @chainable
 *
 * @example
 * <div>
 * <code>
 * // Click and drag the mouse to view the scene from different
angles.
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
```

```
 *
 *   describe('A multicolor torus drawn on a gray background.');
 * }
 *
 * function draw() {
 *   background(200);
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
 *   // Style the torus.
 *   normalMaterial();
 *
 *   // Draw the torus.
 *   torus(30);
 * }
 * </code>
 * </div>
 */
p5.prototype.normalMaterial = function (...args) {
  this._assert3d('normalMaterial');
  p5._validateParameters('normalMaterial', args);
  this._renderer.drawMode = constants.FILL;
  this._renderer._useSpecularMaterial = false;
  this._renderer._useEmissiveMaterial = false;
  this._renderer._useNormalMaterial = true;
  this._renderer.curFillColor = [1, 1, 1, 1];
  this._renderer._setProperty('_doFill', true);
  this.noStroke();
  return this;
};
```

textureWrap Function

1. **Data Referencing Errors**

None found. The function correctly references the wrapX and wrapY parameters to set the texture wrapping modes.

2. **Data Declaration Errors**

None found. The parameters are declared correctly, and default values are handled appropriately.

3. **Computation Errors**

None found. This function does not involve complex computations but effectively sets the texture wrapping mode.

4. **Comparison Errors**

None found. The function uses the parameters directly without any comparison that could lead to errors.

5. **Control Flow Errors**

None found. The function executes straightforwardly, and there are no branching paths that could introduce logic errors.

6. **Interface Errors**

None found. The function manipulates the internal properties of the renderer directly and calls the setWrapMode method on each texture, assuming the textures are properly initialized.

7. **Input/Output Errors**

None found. The function does not perform any I/O operations that could lead to unexpected behavior.

normalMaterial Function

1. **Data Referencing Errors**

None found. All properties referenced are expected to exist on the renderer object.

2. **Data Declaration Errors**

None found. The rest parameter ...args is correctly used to capture any additional arguments, and they are validated.

3. **Computation Errors**

None found. The function sets properties without performing complex calculations.

4. **Comparison Errors**

None found. The function does not contain any conditional logic that could introduce comparison errors.

5. **Control Flow Errors**

None found. The control flow is clear and straightforward, setting properties directly without branching logic.

6. **Interface Errors**

None found. The function calls several methods on the renderer to configure its behavior as a normal material, assuming all necessary methods exist.

7. **Input/Output Errors**

None found. The function does not interact with external input/output.

```
 * @method ambientMaterial
 * @param  {Number} v1  red or hue value in the current
 *                       <a href="#/p5/colorMode">colorMode()</a>.
 * @param  {Number} v2  green or saturation value in the
 *                      current <a
href="#/p5/colorMode">colorMode()</a>.
 * @param  {Number} v3  blue, brightness, or lightness value in the
 *                      current <a
href="#/p5/colorMode">colorMode()</a>.
 * @chainable
 *
 * @example
 * <div>
 * <code>
 * // Click and drag the mouse to view the scene from different
angles.
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
```

```
 *
 *   describe('A magenta cube drawn on a gray background.');
 * }
 *
 * function draw() {
 *   background(200);
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
 *   // Turn on a magenta ambient light.
 *   ambientLight(255, 0, 255);
 *
 *   // Draw the box.
 *   box();
 * }
 * </code>
 * </div>
 *
 * <div>
 * <code>
 * // Click and drag the mouse to view the scene from different
angles.
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe('A purple cube drawn on a gray background.');
 * }
 *
 * function draw() {
 *   background(200);
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
 *   // Turn on a magenta ambient light.
 *   ambientLight(255, 0, 255);
 *
 *   // Add a dark gray ambient material.
 *   ambientMaterial(150);
 *
 *   // Draw the box.
 *   box();
 * }
```

```
 * </code>
 * </div>
 *
 * <div>
 * <code>
 * // Click and drag the mouse to view the scene from different
angles.
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe('A red cube drawn on a gray background.');
 * }
 *
 * function draw() {
 *   background(200);
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
 *   // Turn on a magenta ambient light.
 *   ambientLight(255, 0, 255);
 *
 *   // Add a yellow ambient material using RGB values.
 *   ambientMaterial(255, 255, 0);
 *
 *   // Draw the box.
 *   box();
 * }
 * </code>
 * </div>
 *
 * <div>
 * <code>
 * // Click and drag the mouse to view the scene from different
angles.
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe('A red cube drawn on a gray background.');
 * }
 *
 * function draw() {
 *   background(200);
```

```
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
 *   // Turn on a magenta ambient light.
 *   ambientLight(255, 0, 255);
 *
 *   // Add a yellow ambient material using a p5.Color object.
 *   let c = color(255, 255, 0);
 *   ambientMaterial(c);
 *
 *   // Draw the box.
 *   box();
 * }
 * </code>
 * </div>
 *
 * <div>
 * <code>
 * // Click and drag the mouse to view the scene from different
angles.
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe('A red cube drawn on a gray background.');
 * }
 *
 * function draw() {
 *   background(200);
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
 *   // Turn on a magenta ambient light.
 *   ambientLight(255, 0, 255);
 *
 *   // Add a yellow ambient material using a color string.
 *   ambientMaterial('yellow');
 *
 *   // Draw the box.
 *   box();
 * }
 * </code>
 * </div>
```

```
 *
 * <div>
 * <code>
 * // Click and drag the mouse to view the scene from different
angles.
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe('A yellow cube drawn on a gray background.');
 * }
 *
 * function draw() {
 *   background(200);
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
 *   // Turn on a white ambient light.
 *   ambientLight(255, 255, 255);
 *
 *   // Add a yellow ambient material using a color string.
 *   ambientMaterial('yellow');
 *
 *   // Draw the box.
 *   box();
 * }
 * </code>
 * </div>
 */

/**
 * @method ambientMaterial
 * @param  {Number} gray grayscale value between 0 (black) and 255
(white).
 * @chainable
 */

/**
 * @method ambientMaterial
 * @param  {p5.Color|Number[]|String} color
 *           color as a <a href="#/p5.Color">p5.Color</a> object,
 *            an array of color values, or a CSS string.
 * @chainable
 */
```

```
p5.prototype.ambientMaterial = function (v1, v2, v3) {
  this._assert3d('ambientMaterial');
  p5._validateParameters('ambientMaterial', arguments);

  const color = p5.prototype.color.apply(this, arguments);
  this._renderer._hasSetAmbient = true;
  this._renderer.curAmbientColor = color._array;
  this._renderer._useNormalMaterial = false;
  this._renderer._enableLighting = true;
  this._renderer._setProperty('_doFill', true);
  return this;
};
```

1. **Data Referencing Errors**

   None found. The function references the parameters correctly and accesses properties of the color object that should exist.

2. **Data Declaration Errors**

   None found. The parameters are correctly defined to accept either RGB values or a color object/string.

3. **Computation Errors**

   None found. The function does not perform any complex computations, but it correctly processes the input parameters to create a color.

4. **Comparison Errors**

   None found. The function does not utilize comparison logic that could introduce errors.

5. **Control Flow Errors**

   None found. The flow of execution in the function is straightforward, setting properties on the renderer without branching logic.

6. **Interface Errors**

None found. The function assumes that the renderer has been initialized correctly and can handle the properties being set.

7. **Input/Output Errors**

   None found. The function does not involve any I/O operations that could lead to unexpected behavior.

```
 * @method emissiveMaterial
 * @param  {Number} v1       red or hue value in the current
 *                           <a href="#/p5/colorMode">colorMode()</a>.
 * @param  {Number} v2       green or saturation value in the
 *                           current <a
href="#/p5/colorMode">colorMode()</a>.
 * @param  {Number} v3       blue, brightness, or lightness value in
the
 *                           current <a
href="#/p5/colorMode">colorMode()</a>.
 * @param  {Number} [alpha]  alpha value in the current
 *                           <a href="#/p5/colorMode">colorMode()</a>.
 * @chainable
 *
 * @example
 * <div>
 * <code>
 * // Click and drag the mouse to view the scene from different
angles.
 *
 * function setup() {
 *   createCanvas(100, 100, WEBGL);
 *
 *   describe('A red cube drawn on a gray background.');
 * }
 *
 * function draw() {
 *   background(200);
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
```

```
 *   // Turn on a white ambient light.
 *   ambientLight(255, 255, 255);
 *
 *   // Add a red emissive material using RGB values.
 *   emissiveMaterial(255, 0, 0);
 *
 *   // Draw the box.
 *   box();
 * }
 * </code>
 * </div>
 */

/**
 * @method emissiveMaterial
 * @param  {Number} gray grayscale value between 0 (black) and 255
(white).
 * @chainable
 */

/**
 * @method emissiveMaterial
 * @param  {p5.Color|Number[]|String} color
 *           color as a <a href="#/p5.Color">p5.Color</a> object,
 *            an array of color values, or a CSS string.
 * @chainable
 */
p5.prototype.emissiveMaterial = function (v1, v2, v3, a) {
  this._assert3d('emissiveMaterial');
  p5._validateParameters('emissiveMaterial', arguments);

  const color = p5.prototype.color.apply(this, arguments);
  this._renderer.curEmissiveColor = color._array;
  this._renderer._useEmissiveMaterial = true;
  this._renderer._useNormalMaterial = false;
  this._renderer._enableLighting = true;

  return this;
};
```

1. **Data Referencing Errors**

   None found. The function accurately accesses properties of the color object that are expected to exist.

2. **Data Declaration Errors**

   None found. The parameters are defined to accept RGB values, grayscale values, or color strings correctly.

3. **Computation Errors**

   None found. The function does not perform complex computations; it processes inputs directly to create a color object.

4. **Comparison Errors**

   None found. There are no comparisons in the code that could introduce logical errors.

5. **Control Flow Errors**

   None found. The function has a linear execution flow without branching, making it straightforward.

6. **Interface Errors**

   None found. It assumes the renderer has been correctly initialized and can handle the properties set by this function.

7. **Input/Output Errors**

   None found. The function does not perform I/O operations that could result in errors.

```
 * function draw() {
 *   background(200);
 *
 *   // Enable orbiting with the mouse.
 *   orbitControl();
 *
 *   // Turn on a white point light at the top-right.
 *   pointLight(255, 255, 255, 30, -40, 30);
 *
 *   // Add a glossy green coat if the user has double-clicked.
 *   if (isGlossy === true) {
 *     specularMaterial('#00FF00');
 *     shininess(50);
 *   }
 *
 *   // Style the torus.
 *   noStroke();
 *   fill(255, 0, 0);
 *
 *   // Draw the torus.
 *   torus(30);
 * }
 *
 * // Make the torus glossy when the user double-clicks.
 * function doubleClicked() {
 *   isGlossy = true;
 * }
 * </code>
 * </div>
 */

/**
 * @method specularMaterial
 * @param  {Number}        v1      red or hue value in
 *                                 the current <a
href="#/p5/colorMode">colorMode()</a>.
 * @param  {Number}        v2      green or saturation value
 *                                 in the current <a
href="#/p5/colorMode">colorMode()</a>.
 * @param  {Number}        v3      blue, brightness, or lightness
value
 *                                 in the current <a
href="#/p5/colorMode">colorMode()</a>.
 * @param  {Number}        [alpha]
 * @chainable
```

```
 */

/**
 * @method specularMaterial
 * @param  {p5.Color|Number[]|String} color
 *           color as a <a href="#/p5.Color">p5.Color</a> object,
 *            an array of color values, or a CSS string.
 * @chainable
 */
p5.prototype.specularMaterial = function (v1, v2, v3, alpha) {
  this._assert3d('specularMaterial');
  p5._validateParameters('specularMaterial', arguments);

  const color = p5.prototype.color.apply(this, arguments);
  this._renderer.curSpecularColor = color._array;
  this._renderer._useSpecularMaterial = true;
  this._renderer._useNormalMaterial = false;
  this._renderer._enableLighting = true;

  return this;
};
```

1. **Data Referencing Errors**

   None found. The function correctly accesses the properties of the color object that it generates based on the provided parameters.

2. **Data Declaration Errors**

   None found. The parameters are defined to accept RGB values, alpha values, or color strings properly.

3. **Computation Errors**

   None found. The function does not perform any computations that could lead to errors; it simply applies the input parameters to define a color.

4. **Comparison Errors**

   None found. The function does not use conditional comparisons that could introduce logical errors.

5. **Control Flow Errors**

   None found. The function operates in a straightforward manner without branching, leading to predictable execution.

6. **Interface Errors**

   None found. The function assumes that the renderer and related methods are properly set up and that _assert3d and _validateParameters will handle any issues before the core logic is executed.

7. **Input/Output Errors**

   None found. The function does not engage in any I/O operations that could lead to unexpected behavior.

```
* function draw() {
*   background(200);
*
*   // Turn on a red ambient light.
*   ambientLight(255, 0, 0);
*
*   // Get the mouse's coordinates.
*   let mx = mouseX - 50;
*   let my = mouseY - 50;
*
*   // Turn on a white point light that follows the mouse.
*   pointLight(255, 255, 255, mx, my, 50);
*
*   // Style the sphere.
*   noStroke();
*
```

```
 *   // Add a specular material with a grayscale value.
 *   specularMaterial(255);
 *
 *   // Draw the left sphere with low shininess.
 *   translate(-25, 0, 0);
 *   shininess(10);
 *   sphere(20);
 *
 *   // Draw the right sphere with high shininess.
 *   translate(50, 0, 0);
 *   shininess(100);
 *   sphere(20);
 * }
 * </code>
 * </div>
 */
p5.prototype.shininess = function (shine) {
  this._assert3d('shininess');
  p5._validateParameters('shininess', arguments);

  if (shine < 1) {
    shine = 1;
  }
  this._renderer._useShininess = shine;
  return this;
};
```

1. **Data Referencing Errors**

   None found. The function correctly references the properties needed to set the shininess of the material in the rendering context.

2. **Data Declaration Errors**

   None found. The parameter shine is appropriately declared, and the function does not have issues with data types.

3. **Computation Errors**

   None found. The function correctly applies a minimum limit to the shine value, ensuring it never falls below 1, which is sensible for maintaining valid shininess in a 3D rendering context.

4. **Comparison Errors**

   None found. The comparison to set a minimum value for shine is straightforward and effective, preventing negative or zero shininess.

5. **Control Flow Errors**

   None found. The flow of the function is linear and predictable; it performs a single check before applying the property.

6. **Interface Errors**

   None found. The function assumes the renderer is correctly set up and that the validation checks for parameters (_validateParameters) will properly catch any erroneous inputs before execution.

7. **Input/Output Errors**

   None found. The function does not handle any input/output operations that could lead to errors.

```
/**
 * @private blends colors according to color components.
 * If alpha value is less than 1, or non-standard blendMode
 * we need to enable blending on our gl context.
 * @param  {Number[]} color The currently set color, with values in 0-
1 range
 * @param  {Boolean} [hasTransparency] Whether the shape being drawn
has other
 * transparency internally, e.g. via vertex colors
 * @return {Number[]}  Normalized numbers array
 */
p5.RendererGL.prototype._applyColorBlend = function(colors,
hasTransparency) {
  const gl = this.GL;

  const isTexture = this.drawMode === constants.TEXTURE;
  const doBlend =
```

```
    hasTransparency ||
    this.userFillShader ||
    this.userStrokeShader ||
    this.userPointShader ||
    isTexture ||
    this.curBlendMode !== constants.BLEND ||
    colors[colors.length - 1] < 1.0 ||
    this._isErasing;

  if (doBlend !== this._isBlending) {
    if (
      doBlend ||
      (this.curBlendMode !== constants.BLEND &&
        this.curBlendMode !== constants.ADD)
    ) {
      gl.enable(gl.BLEND);
    } else {
      gl.disable(gl.BLEND);
    }
    gl.depthMask(true);
    this._isBlending = doBlend;
  }
  this._applyBlendMode();
  return colors;
};

/**
 * @private sets blending in gl context to curBlendMode
 * @param  {Number[]} color [description]
 * @return {Number[]}  Normalized numbers array
 */
p5.RendererGL.prototype._applyBlendMode = function () {
  if (this._cachedBlendMode === this.curBlendMode) {
    return;
  }
  const gl = this.GL;
  switch (this.curBlendMode) {
    case constants.BLEND:
      gl.blendEquation(gl.FUNC_ADD);
      gl.blendFunc(gl.ONE, gl.ONE_MINUS_SRC_ALPHA);
      break;
    case constants.ADD:
      gl.blendEquation(gl.FUNC_ADD);
      gl.blendFunc(gl.ONE, gl.ONE);
      break;
```

```
    case constants.REMOVE:
      gl.blendEquation(gl.FUNC_ADD);
      gl.blendFunc(gl.ZERO, gl.ONE_MINUS_SRC_ALPHA);
      break;
    case constants.MULTIPLY:
      gl.blendEquation(gl.FUNC_ADD);
      gl.blendFunc(gl.DST_COLOR, gl.ONE_MINUS_SRC_ALPHA);
      break;
    case constants.SCREEN:
      gl.blendEquation(gl.FUNC_ADD);
      gl.blendFunc(gl.ONE, gl.ONE_MINUS_SRC_COLOR);
      break;
    case constants.EXCLUSION:
      gl.blendEquationSeparate(gl.FUNC_ADD, gl.FUNC_ADD);
      gl.blendFuncSeparate(
        gl.ONE_MINUS_DST_COLOR,
        gl.ONE_MINUS_SRC_COLOR,
        gl.ONE,
        gl.ONE
      );
      break;
    case constants.REPLACE:
      gl.blendEquation(gl.FUNC_ADD);
      gl.blendFunc(gl.ONE, gl.ZERO);
      break;
    case constants.SUBTRACT:
      gl.blendEquationSeparate(gl.FUNC_REVERSE_SUBTRACT, gl.FUNC_ADD);
      gl.blendFuncSeparate(gl.ONE, gl.ONE, gl.ONE,
gl.ONE_MINUS_SRC_ALPHA);
      break;
    case constants.DARKEST:
      if (this.blendExt) {
        gl.blendEquationSeparate(
          this.blendExt.MIN || this.blendExt.MIN_EXT,
          gl.FUNC_ADD
        );
        gl.blendFuncSeparate(gl.ONE, gl.ONE, gl.ONE, gl.ONE);
      } else {
        console.warn(
          'blendMode(DARKEST) does not work in your browser in WEBGL
mode.'
        );
      }
      break;
    case constants.LIGHTEST:
```

```
      if (this.blendExt) {
        gl.blendEquationSeparate(
          this.blendExt.MAX || this.blendExt.MAX_EXT,
          gl.FUNC_ADD
        );
        gl.blendFuncSeparate(gl.ONE, gl.ONE, gl.ONE, gl.ONE);
      } else {
        console.warn(
          'blendMode(LIGHTEST) does not work in your browser in WEBGL
mode.'
        );
      }
      break;
    default:
      console.error(
        'Oops! Somehow RendererGL set curBlendMode to an unsupported
mode.'
      );
      break;
  }
  if (!this._isErasing) {
    this._cachedBlendMode = this.curBlendMode;
  }
};

export default p5;
```

applyColorBlend Function

1. **Data Referencing Errors**

None found. The function references necessary properties like GL, drawMode, curBlendMode, and colors effectively.

2. **Data Declaration Errors**

None found. All parameters are declared properly, with colors expected to be an array of numbers.

3. **Computation Errors**

None found. The logic for determining whether blending should occur is clearly defined and considers multiple factors.

4. **Comparison Errors**

None found. The checks for transparency and blend mode comparisons are straightforward and effective.

5. **Control Flow Errors**

None found. The conditional checks and branching (enabling/disabling blending) are well-structured and logical.

6. **Interface Errors**

None found. The method relies on the rendering context (GL) and correctly interacts with it to set blending states.

7. **Input/Output Errors**

None found. The function outputs the modified color array as expected.

applyBlendMode Function

1. **Data Referencing Errors**

None found. The function references GL and curBlendMode effectively for setting blend modes.

2. **Data Declaration Errors**

None found. The method does not introduce new variables that could lead to confusion.

3. **Computation Errors**

None found. The blend modes are set according to the defined constants, and each case is handled correctly.

4. **Comparison Errors**

None found. The comparison of _cachedBlendMode to curBlendMode to prevent redundant state changes is efficient.

5. **Control Flow Errors**

None found. The switch statement effectively directs flow based on curBlendMode, and all cases appear to be covered.

6. **Interface Errors**

None found. The function interacts directly with the GL context to set blending modes, which is standard practice.

7. **Input/Output Errors**

None found. The function does not involve input/output operations that could introduce errors.

II. CODE DEBUGGING

Debugging is a systematic process involving the localization, analysis, and removal of suspected errors in the provided Java code. It is essential for ensuring code quality and functionality.

➔ Armstrong.txt

1. **Data Reference/Initialization Errors**:
    - In the while loop, the line `remainder = num / 10;` is intended to extract the last digit. However, the correct operation should be `remainder = num % 10;` to properly obtain the last digit instead of the quotient.
2. **Computation Errors**:
    - The line `remainder = num / 10;` is incorrect for the intended calculation. It should be revised to `remainder = num % 10;`, as this operation correctly retrieves the last digit of the number.
3. **Control Flow/Comparison Errors**:
    - While there doesn't appear to be an issue with control flow, the comparison `check == n` functions correctly, ensuring the logic is intact.
4. **I/O Errors**:
    - The input and output operations are functioning correctly; the program successfully prints whether the number is an Armstrong number based on the calculated results.

➔ GCD and LCM

1. **Data Reference/Initialization Errors**:
    - In the GCD calculation, the initialization of variables `a` and `b` is based on conditions that are not correct, which can lead to erroneous calculations.
2. **Control Flow/Comparison Errors**:

- The logic in the LCM calculation has control flow issues, specifically where conditions for determining the LCM are incorrectly compared, potentially resulting in an infinite loop or incorrect output.

3. **I/O Errors**:
   - There are no input/output errors; the program correctly accepts user input and provides accurate outputs for both GCD and LCM calculations.

➔ Knapsack

- One error has been identified in the program's logic. To resolve this, set a breakpoint at `int option1 = opt[n][w];` to ensure that the variables `n` and `w` are utilized correctly without unintended increments affecting the output.

➔ Magic Number

- Two errors have been pinpointed in the program's logic. To address these, set a breakpoint at the start of the inner while loop to confirm its execution. Additionally, using breakpoints will help track the values of `num` and `s` during execution to identify any discrepancies.

➔ Merge Sort

- Multiple errors are present within the program's implementation. Setting breakpoints to observe the values of `left`, `right`, and `array` during execution will be beneficial. It's also crucial to monitor the values of `i1` and `i2` inside the merge method to ensure proper functionality.

➔ Multiply Matrices

- Multiple errors have been detected in the program. To rectify these, set breakpoints to review the values of `c`, `d`, `k`, and `sum` during execution, focusing particularly on the nested loops where matrix multiplication occurs.

➔ Quadratic Probing

- Three errors have been identified in the program's logic. To fix these errors, set breakpoints and step through the code while examining variables such as `i`, `h`, `tmp1`, and `tmp2`. It is important to scrutinize the logic in the insert, remove, and get methods for any inconsistencies.

➔ Sorting Array

- Two errors are present within the program as noted earlier. To resolve these, set breakpoints and step through the code, paying attention to the class name, loop conditions, and any unnecessary semicolons that may be causing issues.

➔ Stack Implementation

- Three errors have been identified in the program's functionality. To address these, set breakpoints and step through the code while focusing on the push, pop, and display methods. Make sure to correct the push and display methods, and don’t forget to implement the missing pop method to complete the stack functionality.

➔ Tower of Hanoi

- One error has been identified in the implementation of this algorithm. To rectify this, replace the line: `doTowers(topN ++, inter--, from+1, to+1);` with the corrected version: `doTowers(topN - 1, inter, from, to);` to ensure proper execution.