EXERCISE - 1

AIM : To implement a line-drawing algorithm (e.g., DDA or Bresenham's algorithm) in C to draw a straight line between two points**.**

Procedure (Using Bresenham's Algorithm)

1. Input:
   - Two endpoints (x1,y1) and (x2,y2).
2. Calculate the differences:
   - Δx=x2−x1
   - Δy=y2−y1
3. Determine the decision parameters:
   - Set p = 2Δy - Δx for the initial decision variable.
4. Iteratively plot points:
   - Start from the first endpoint and move toward the second endpoint.
   - Based on the decision variable p, determine whether to increment the y-coordinate.
5. Repeat until the second endpoint is reached.

SAMPLE CODE:

```
#include <stdio.h>
#include <graphics.h>

void bresenhamLine(int x1, int y1, int x2, int y2) {
   int dx = x2 - x1;
   int dy = y2 - y1;

   int p = 2 * dy - dx; // Initial decision parameter
   int x = x1, y = y1;

   // Plot the first point
   putpixel(x, y, WHITE);

   // Iterate through the points
```

```
    while (x < x2) {
        x++;
        if (p < 0) {
            p += 2 * dy; // Mid-point below the line
        } else {
            y++;
            p += 2 * (dy - dx); // Mid-point above or on the line
        }
        putpixel(x, y, WHITE); // Plot the next point
    }
}

int main() {
    int gd = DETECT, gm;
    int x1, y1, x2, y2;

    // Initialize the graphics system
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    // Input endpoints
    printf("Enter the coordinates of the first point (x1, y1): ");
    scanf("%d %d", &x1, &y1);
    printf("Enter the coordinates of the second point (x2, y2): ");
    scanf("%d %d", &x2, &y2);

    // Draw the line using Bresenham's algorithm
    bresenhamLine(x1, y1, x2, y2);

    // Wait for user input to close the graphics window
    getch();
    closegraph();

    return 0;
}
```

EXERCISE - 2

AIM : To implement Bresenham's circle-drawing algorithm in C to draw a circle on a graphical interface..

Procedure (Using Bresenham’s Algorithm)

1. Input the radius and center coordinates (xc,yc).
2. Initialize parameters:
    - Start from the point (0,r)on the circle.
    - Compute the initial decision parameter:
      $p=3-2r$.
3. Plot initial points:
    - Using symmetry, plot points in all 8 octants of the circle based on (x,y).
4. Update decision parameter:
    - If $p<0$, the next point is $(x+1,y)$. Update p as:
      $p=p+4x+6$
    - Otherwise, the next point is $(x+1,y-1)$. Update p as:
      $p=p+4(x-y)+10$.
5. Repeat until $x \geq y$:
    - Continue plotting points in all octants.

SAMPLE CODE:

```
#include <stdio.h>
#include <graphics.h>

void plotCirclePoints(int xc, int yc, int x, int y) {
    putpixel(xc + x, yc + y, WHITE); // Octant 1
    putpixel(xc - x, yc + y, WHITE); // Octant 2
    putpixel(xc + x, yc - y, WHITE); // Octant 3
    putpixel(xc - x, yc - y, WHITE); // Octant 4
    putpixel(xc + y, yc + x, WHITE); // Octant 5
    putpixel(xc - y, yc + x, WHITE); // Octant 6
    putpixel(xc + y, yc - x, WHITE); // Octant 7
    putpixel(xc - y, yc - x, WHITE); // Octant 8
}
```

```
void bresenhamCircle(int xc, int yc, int r) {
    int x = 0, y = r;
    int p = 3 - 2 * r; // Initial decision parameter

    plotCirclePoints(xc, yc, x, y);

    while (x <= y) {
        x++;
        if (p < 0) {
            p = p + 4 * x + 6; // Mid-point inside or on the perimeter
        } else {
            y--;
            p = p + 4 * (x - y) + 10; // Mid-point outside the perimeter
        }
        plotCirclePoints(xc, yc, x, y);
    }
}

int main() {
    int gd = DETECT, gm;
    int xc, yc, r;

    // Initialize graphics mode
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    // Input center and radius
    printf("Enter the center of the circle (xc, yc): ");
    scanf("%d %d", &xc, &yc);
    printf("Enter the radius of the circle: ");
    scanf("%d", &r);

    // Draw the circle
    bresenhamCircle(xc, yc, r);

    // Wait for user input and close the graphics window
    getch();
    closegraph();

    return 0;
}
```

EXERCISE - 3

AIM : To implement 2D transformations such as translation, scaling, and rotation on a 2D object using C programming and a graphics library.

Procedure (Using Bresenham's Algorithm)

2D transformations are used to manipulate objects in a two-dimensional plane. Common transformations include:

1. **Translation:** Moves an object from one location to another by adding offsets to the coordinates.

   $x'=x+tx, y'=y+ty$

2. **Scaling:** Resizes an object by scaling factors Sx and Sy.

   $x'=x\cdot Sx$, $y'=y\cdot Sy$

3. **Rotation:** Rotates an object around the origin by an angle θ

   $x'=x\cdot\cos\theta-y\cdot\sin\theta$, $y'=x\cdot\sin\theta+y\cdot\cos\theta$

Step 1 : Input the object coordinates:
Define the vertices of the object (e.g., a triangle or square).

Step 2 :   Choose the transformation type:
Translation, scaling, or rotation.

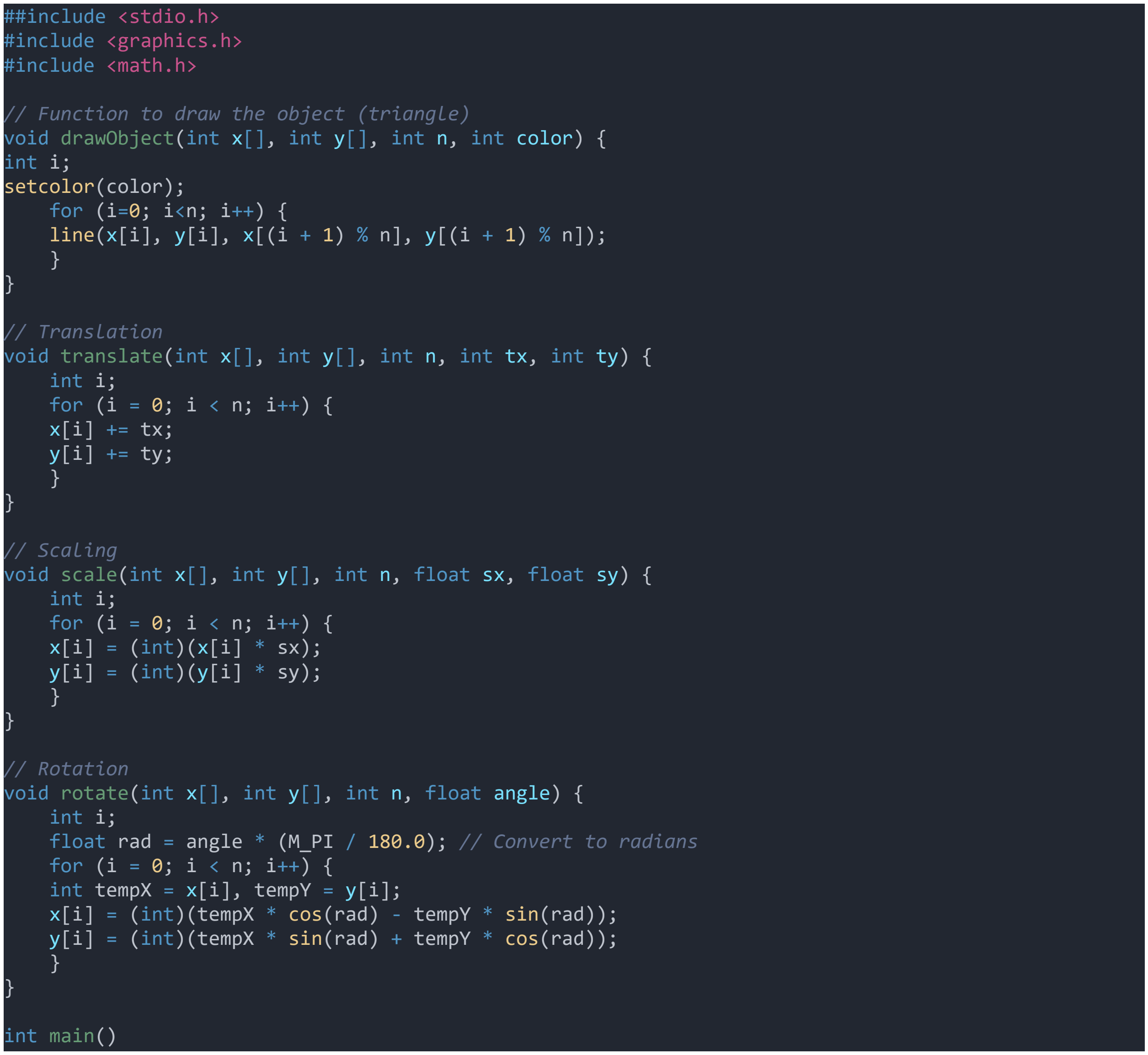
Step 3 : Apply the transformation:

Use the appropriate formulas to calculate the transformed coordinates.

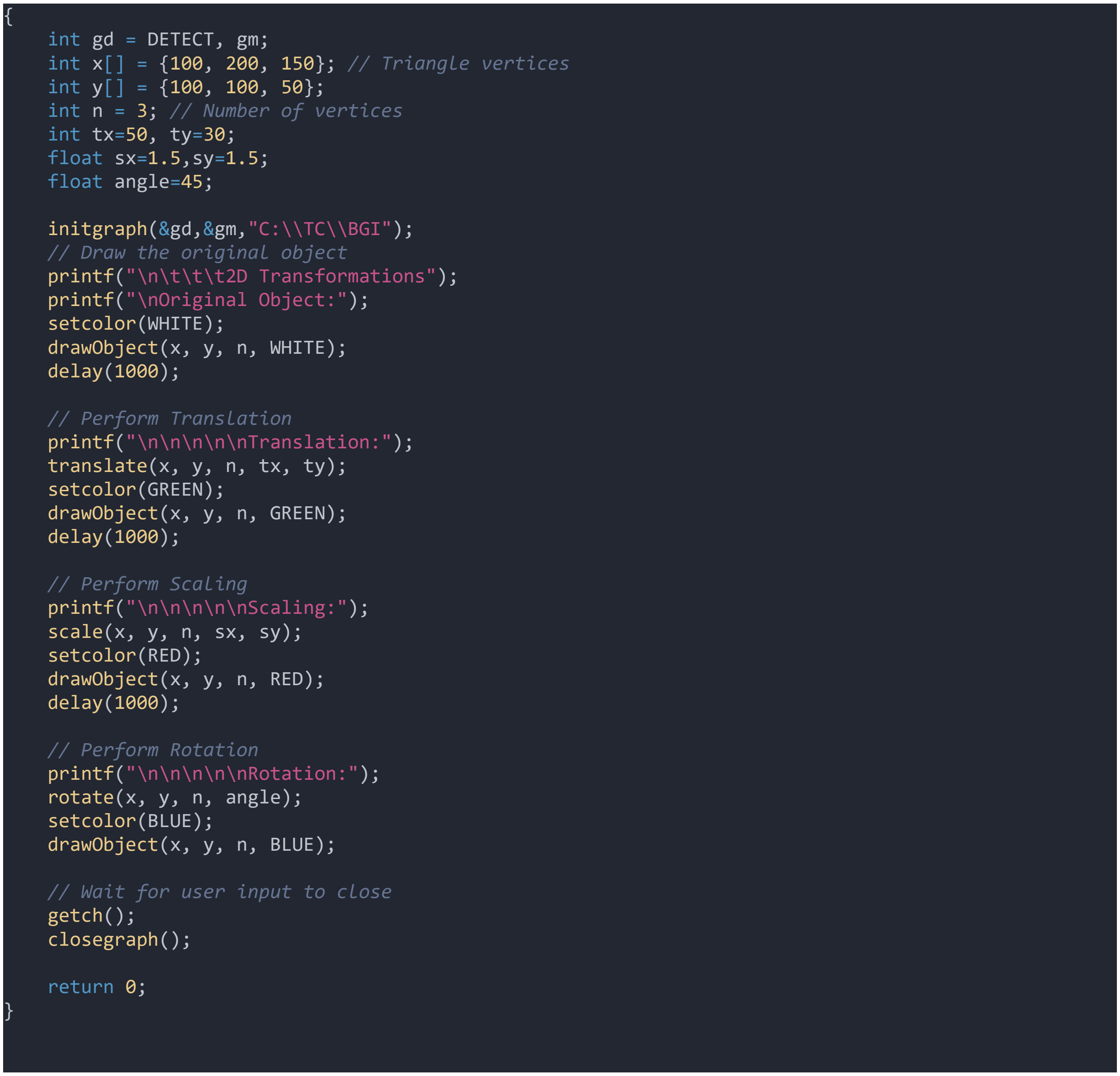
Step 4 : Display the results:

Render the original and transformed objects on the screen.

SAMPLE CODE:



```c
##include <stdio.h>
#include <graphics.h>
#include <math.h>

// Function to draw the object (triangle)
void drawObject(int x[], int y[], int n, int color) {
int i;
setcolor(color);
    for (i=0; i<n; i++) {
    line(x[i], y[i], x[(i + 1) % n], y[(i + 1) % n]);
    }
}

// Translation
void translate(int x[], int y[], int n, int tx, int ty) {
    int i;
    for (i = 0; i < n; i++) {
    x[i] += tx;
    y[i] += ty;
    }
}

// Scaling
void scale(int x[], int y[], int n, float sx, float sy) {
    int i;
    for (i = 0; i < n; i++) {
    x[i] = (int)(x[i] * sx);
    y[i] = (int)(y[i] * sy);
    }
}

// Rotation
void rotate(int x[], int y[], int n, float angle) {
    int i;
    float rad = angle * (M_PI / 180.0); // Convert to radians
    for (i = 0; i < n; i++) {
    int tempX = x[i], tempY = y[i];
    x[i] = (int)(tempX * cos(rad) - tempY * sin(rad));
    y[i] = (int)(tempX * sin(rad) + tempY * cos(rad));
    }
}

int main()
```



```c
{
    int gd = DETECT, gm;
    int x[] = {100, 200, 150}; // Triangle vertices
    int y[] = {100, 100, 50};
    int n = 3; // Number of vertices
    int tx=50, ty=30;
    float sx=1.5,sy=1.5;
    float angle=45;

    initgraph(&gd,&gm,"C:\\TC\\BGI");
    // Draw the original object
    printf("\n\t\t\t2D Transformations");
    printf("\nOriginal Object:");
    setcolor(WHITE);
    drawObject(x, y, n, WHITE);
    delay(1000);

    // Perform Translation
    printf("\n\n\n\n\nTranslation:");
    translate(x, y, n, tx, ty);
    setcolor(GREEN);
    drawObject(x, y, n, GREEN);
    delay(1000);

    // Perform Scaling
    printf("\n\n\n\n\nScaling:");
    scale(x, y, n, sx, sy);
    setcolor(RED);
    drawObject(x, y, n, RED);
    delay(1000);

    // Perform Rotation
    printf("\n\n\n\n\nRotation:");
    rotate(x, y, n, angle);
    setcolor(BLUE);
    drawObject(x, y, n, BLUE);

    // Wait for user input to close
    getch();
    closegraph();

    return 0;
}
```

EXERCISE - 4

AIM : To implement 3D transformation operations (translation, scaling, and rotation) on an object and display the transformed object using graphical techniques.

PROCEDURE:

1. **Input:**
   - o The 3D coordinates of the object.
   - o Transformation parameters (e.g., translation distances, scaling factors, or rotation angles).
2. **Apply Transformation Matrix:**
   - o For each transformation, multiply the object's 3D coordinates with the appropriate transformation matrix:
     - Translation matrix:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

     - Scaling matrix:

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

     - Rotation matrices (e.g., rotation about the z-axis):

   - o

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3 **Transform the Object:**

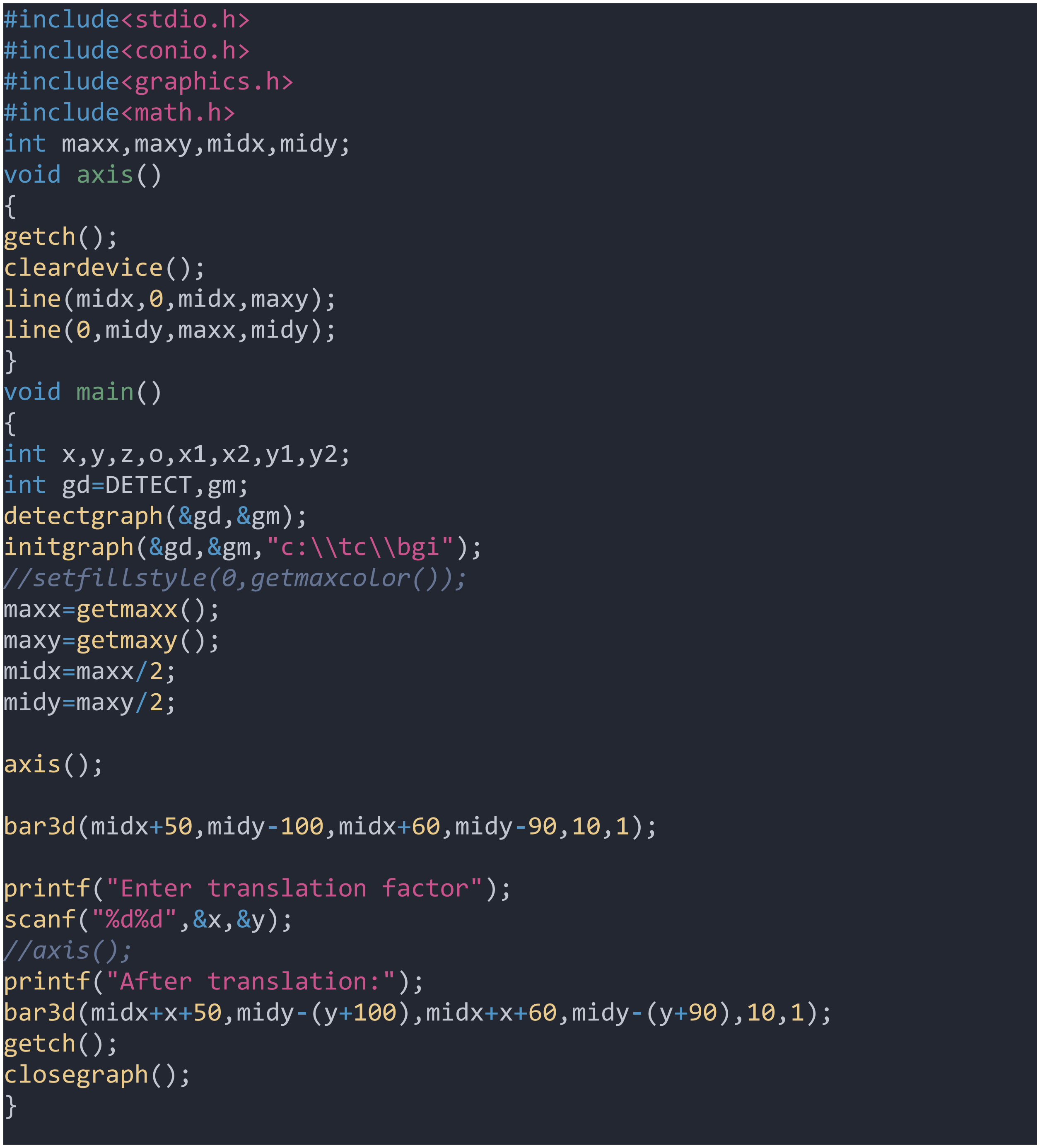
- Multiply the transformation matrix with each point of the object to compute the transformed coordinates.

4 **Output:**

- Display the original and transformed object using a graphical interface.

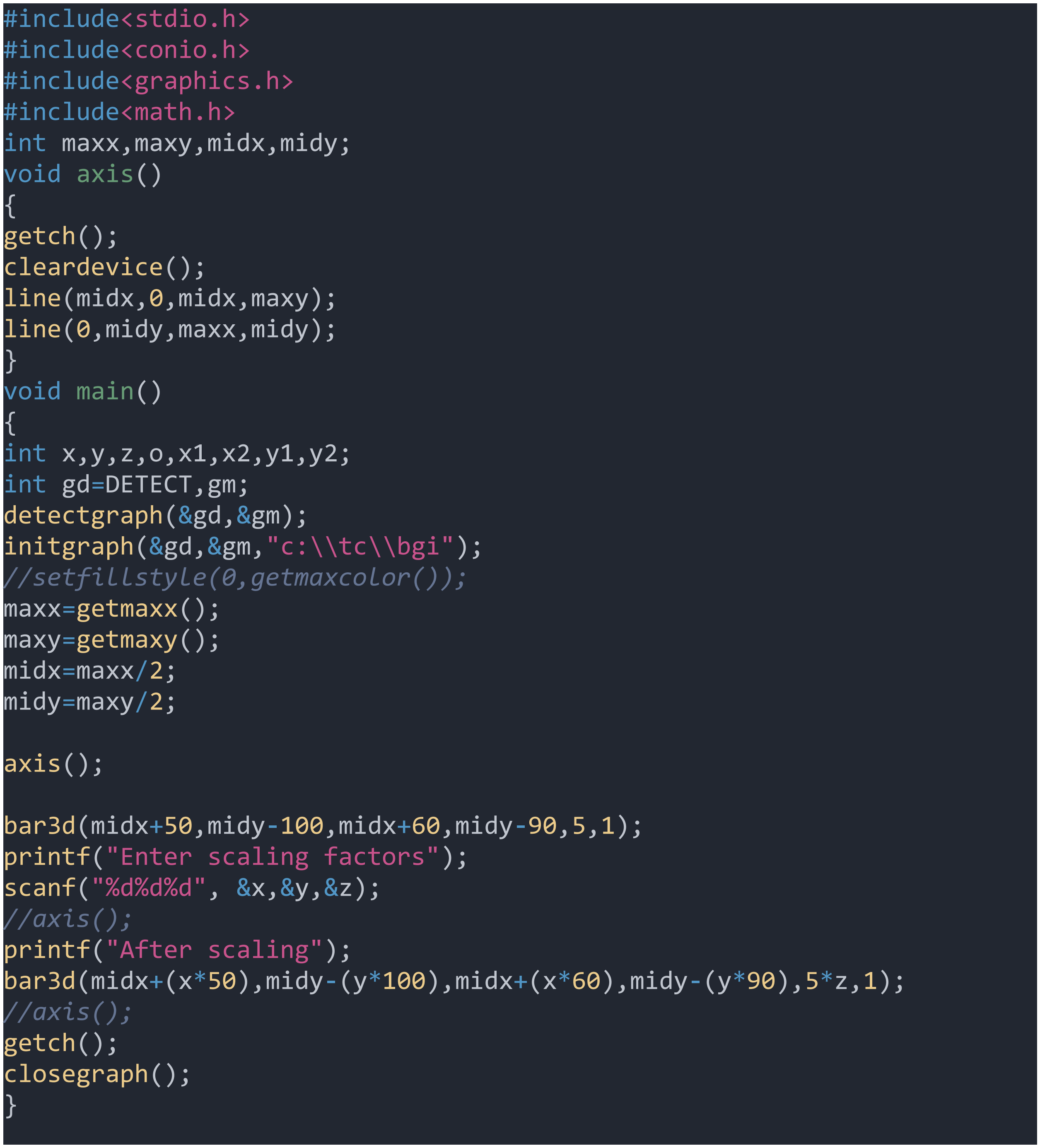
SAMPLE CODE:

<u>3D Translation</u>:



```c
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int maxx,maxy,midx,midy;
void axis()
{
getch();
cleardevice();
line(midx,0,midx,maxy);
line(0,midy,maxx,midy);
}
void main()
{
int x,y,z,o,x1,x2,y1,y2;
int gd=DETECT,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\tc\\bgi");
//setfillstyle(0,getmaxcolor());
maxx=getmaxx();
maxy=getmaxy();
midx=maxx/2;
midy=maxy/2;

axis();

bar3d(midx+50,midy-100,midx+60,midy-90,10,1);

printf("Enter translation factor");
scanf("%d%d",&x,&y);
//axis();
printf("After translation:");
bar3d(midx+x+50,midy-(y+100),midx+x+60,midy-(y+90),10,1);
getch();
closegraph();
}
```

3D Scaling:



```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int maxx,maxy,midx,midy;
void axis()
{
getch();
cleardevice();
line(midx,0,midx,maxy);
line(0,midy,maxx,midy);
}
void main()
{
int x,y,z,o,x1,x2,y1,y2;
int gd=DETECT,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\tc\\bgi");
//setfillstyle(0,getmaxcolor());
maxx=getmaxx();
maxy=getmaxy();
midx=maxx/2;
midy=maxy/2;

axis();

bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
printf("Enter scaling factors");
scanf("%d%d%d", &x,&y,&z);
//axis();
printf("After scaling");
bar3d(midx+(x*50),midy-(y*100),midx+(x*60),midy-(y*90),5*z,1);
//axis();
getch();
closegraph();
}
```

3D Rotation:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int maxx,maxy,midx,midy;
void axis()
{
getch();
cleardevice();
line(midx,0,midx,maxy);
line(0,midy,maxx,midy);
}
void main()
{
int x,y,z,o,x1,x2,y1,y2;
int gd=DETECT,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\tc\\bgi");
//setfillstyle(0,getmaxcolor());
maxx=getmaxx();
maxy=getmaxy();
midx=maxx/2;
midy=maxy/2;
axis();
bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
printf("Enter rotating angle");
scanf("%d",&o);
 x1=50*cos(o*3.14/180)-100*sin(o*3.14/180);
 y1=50*sin(o*3.14/180)+100*cos(o*3.14/180);
 x2=60*cos(o*3.14/180)-90*sin(o*3.14/180);
 y2=60*sin(o*3.14/180)+90*cos(o*3.14/180);
 axis();
printf("After rotation about z axis");
bar3d(midx+x1,midy-y1,midx+x2,midy-y2,5,1);
axis();
printf("After rotation  about x axis");
bar3d(midx+50,midy-x1,midx+60,midy-x2,5,1);
axis();
printf("After rotation about yaxis");
bar3d(midx+x1,midy-100,midx+x2,midy-90,5,1);
getch();
closegraph();
}
```

EXERCISE 5

AIM : To perform video editing tasks such as trimming, adding transitions, applying effects, and exporting the final video using Blender, a free and open-source video editing software.

PROCEDURE:

1. Install Blender

- Go to the official site: https://www.blender.org.
- Download the latest stable version (Windows, Mac, or Linux).
- Run the installer and complete setup (keep default options).
- Open Blender after installation.

---

2. Switch to Video Editing Workspace

- By default, Blender opens in 3D View.
- From the top menu, choose **File → New → Video Editing**.
- Now you're in the **Video Sequence Editor (VSE)**.

---

3. Set Project Properties

- In the right panel, click **Output Properties (printer icon)**.
- Set **Resolution** → 1920 × 1080 (Full HD).
- Set **Frame Rate** → 24, 30, or 60 fps (depending on your video).
- Set **Output Folder** where the final video will be saved.

---

4. Import Media (Video, Images, Audio)

- In the timeline, press **Shift + A** → choose **Movie, Image, or Sound**.
- Browse your files and import them.
- Your clips appear as “strips” on the timeline.

---

5. Arrange and Edit Clips

- Drag strips to change their order or timing.
- Trim clips by dragging their edges.
- Use **K (Knife Tool)** to cut clips.

---

6. Add Transitions and Effects

- Overlap two clips → press **Shift + A** → **Effect Strip** → **Cross / Wipe / Fade**.
- For color correction, speed changes, or overlays → add **Effect Strips**.
- To add text: **Shift + A** → **Text**, then edit in the right panel.

---

7. Add and Adjust Audio

- Import music/voice-over using **Shift + A** → **Sound**.
- Adjust volume in the **Strip Properties panel**.
- Sync audio with video by moving it along the timeline.

---

8. Preview the Video

- Press **Spacebar** to play the sequence.
- Make adjustments until everything looks smooth.

---

9. Export the Final Video

- Go to **Properties** → **Output Properties** → **File Format = FFmpeg Video**.
- Choose **Container = MPEG-4** (MP4).
- Set **Audio Codec = AAC**.
- Click **Render** → **Render Animation (Ctrl + F12)**.
- Blender saves the video in your chosen output folder.

---

10. Save the Project File

- Go to **File** → **Save As** and save the project as a **.blend file**.
- This keeps all your editing work for future changes.

EXERCISE 6

AIM : To create a professional-quality movie clip using Blender, an open-source video editing software.

PROCEDURE:

1. Install and Open Blender

- Download Blender from blender.org and install.
- Open Blender → by default, you'll see the **3D Viewport**.

---

2. Set Up Your Project

- Go to **Properties** → **Output Properties (printer icon)**.
- Choose resolution (e.g., 1920×1080), frame rate (24/30 fps).
- Set start & end frames (e.g., 1–250 frames).

---

3. Create or Import Objects

- Use Blender’s tools to **add 3D objects**:
  - Press **Shift + A → Mesh → Cube, Sphere, etc.**
- Or import models (FBX, OBJ, etc.) if you already have assets.

---

4. Design the Scene

- Position objects in the 3D space.
- Add **materials, textures, and lighting**.
- Place a **camera** (Shift + A → Camera → position using “Numpad 0” to view through it).

---

5. Animate the Objects

- Select an object → move to a frame on the timeline.
- Press **I** (Insert Keyframe) → choose **Location/Rotation/Scale**.
- Move to another frame → change position/rotation → insert another keyframe.
- Blender will **animate between frames** (interpolation).

---

6. Add Audio (Optional)

- Switch to **Video Sequencer** or Timeline → add sound files.
- Sync audio with animation if needed.

---

7. Preview the Animation

- Press **Spacebar** to play the timeline.
- Use **Viewport Shading → Rendered mode** to preview how it looks.

---

8. Render the Movie Clip

- Go to **Properties → Output Properties → File Format → FFmpeg Video (MP4)**.
- Choose output folder and name.

- Press **Render → Render Animation (Ctrl + F12)**.
- Blender will render frames and save them as a **movie clip**.

---

9. Save Your Project

- Save your Blender file (**File → Save As → .blend**) for future editing.
- Your rendered movie clip (MP4) will be in the output folder.

EXERCISE 7

AIM : Create a rotating 3D cube in Blender, animate it, and then import the animation into OpenShot to finalize the video.

PROCEDURE:

Step 1: Create the 3D Model

1. **Select a 3D Modeling Tool**: You need software to create the 3D models. Blender is a free and open-source option that is widely used.
   - **Download Blender** from blender.org.
   - **Install Blender** and launch the program.
2. **Modeling the Object**:
   - Open Blender and start a new project.
   - Use the available tools (such as adding meshes, extruding, scaling, and rotating) to create the 3D object.
   - Adjust the geometry, texture, and materials to suit your project.

3. **Texturing and Materials** (Optional):
   - You can apply textures to your models by using the UV Mapping feature in Blender.
   - Add materials (colors, reflective properties, etc.) through the "Material Properties" tab.
4. **Lighting and Camera Setup**:
   - Set up lighting sources to illuminate your model and define how it looks under different lighting conditions.
   - Place the camera to define the perspective in which your model will be viewed.

Step 2: Animate the 3D Model (Optional)

1. **Add Animations in Blender**:
   - If you want to animate the model, you can use Blender's timeline and keyframe system.
   - Keyframes allow you to change properties (position, rotation, scale) over time.
   - You can also animate textures or materials for added realism.
2. **Rendering the Animation**:
   - Once your model is animated, go to the **Render** tab.
   - Choose the rendering engine (Eevee or Cycles for Blender).
   - Set the output resolution, frame rate, and format (such as MP4 or MOV).
   - Render the animation to an output folder.

Step 3: Import the 3D Animation into OpenShot

1. **Install OpenShot**:
   - Download and install OpenShot from openshot.org.
   - Open the OpenShot software.
2. **Import the 3D Animation Video**:
   - Click on "File" > "Import Files" to import the rendered 3D animation file (e.g., MP4).
   - Drag the animation into the timeline to start working with it.

3. **Editing the 3D Animation in OpenShot**:
   - You can use OpenShot to cut, trim, add transitions, and apply effects to the 3D animation video.
   - You can also overlay other media such as background music or additional footage.
4. **Adding Titles and Text**:
   - Use the “Title” menu in OpenShot to add text and titles to your video.
   - Customize the font, color, and position of the text.
5. **Export the Final Video**:
   - Once your video editing is complete, go to “File” > “Export Project” to export the final video.
   - Choose your desired resolution and format for the final output.