Computer Graphics Practicals

Assignment 6

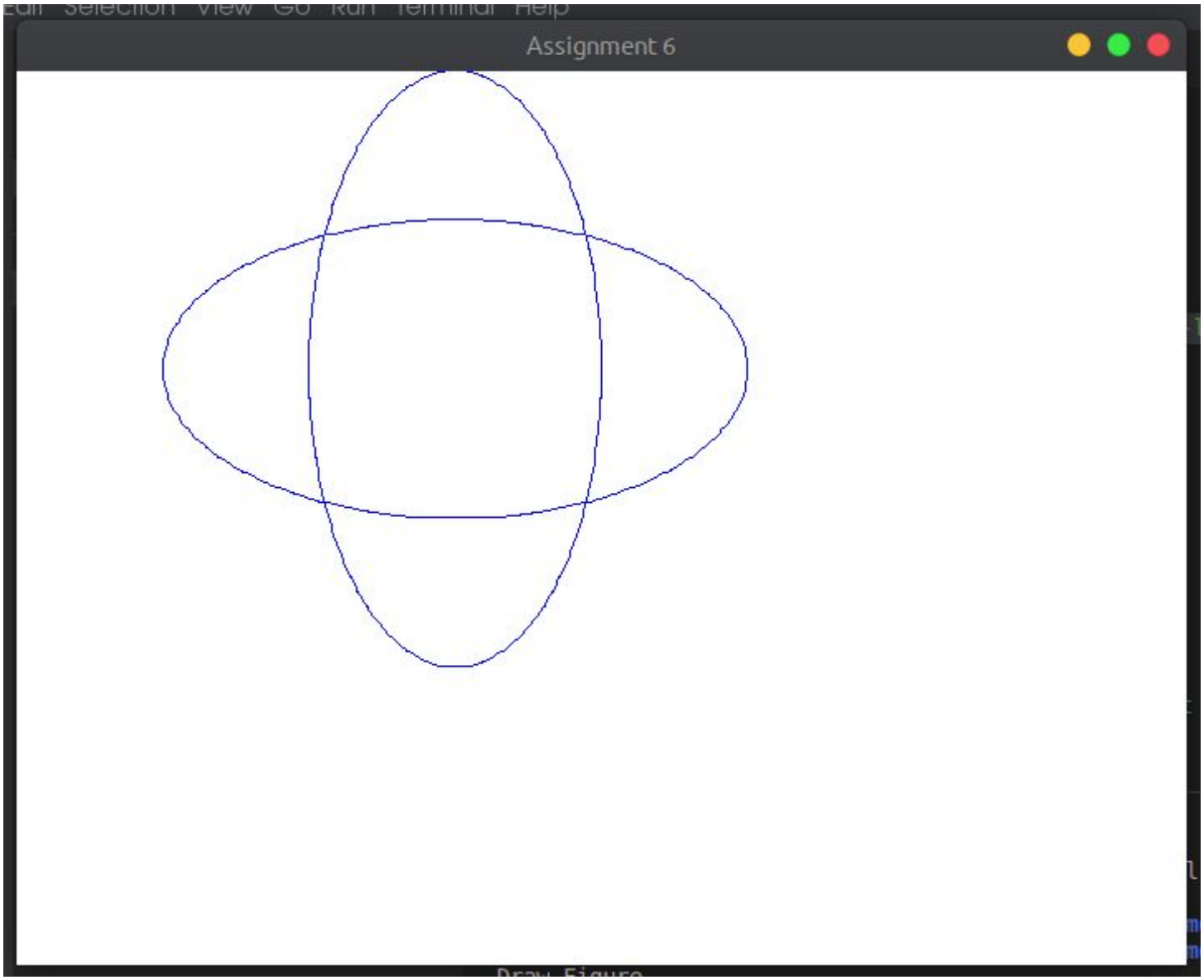
U18CO081
Krunal Rank

Write a program to draw the following figure:-

```
#include <bits/stdc++.h>
#include <GL/glut.h>

using namespace std;

// To Compile:-
// g++ -pthread 1.cpp -lglfw3 -lGLEW -lGLU -lGL -lXrandr -lXxf86vm -lXi -lXinerama
-lX11 -lrt -ldl -lglut

int rx,ry,cx,cy;

void plot(int x, int y)
{
   glBegin(GL_POINTS);
   glVertex2i(x,y);
   glEnd();
}


void midpoint_ellipse_util(int rx,int ry,int cx,int cy){
   float dx, dy, p1, p2, x, y;
   x = 0;
   y = ry;
    p1 = (ry * ry) - (rx * rx * ry) +
                    (0.25 * rx * rx);
   dx = 2 * ry * ry * x;
   dy = 2 * rx * rx * y;
    while (dx < dy)
   {
       plot(cx+x,cy+y);
       plot(cx-x,cy+y);
       plot(cx+x,cy-y);
       plot(cx-x,cy-y);
       if (p1 < 0)
       {
           x++;
           dx = dx + (2 * ry * ry);
           p1 = p1 + dx + (ry * ry);
       }
```

```
        else
        {
            x++;
            y--;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            p1 = p1 + dx - dy + (ry * ry);
        }
    }
     p2 = ((ry * ry) * ((x + 0.5) * (x + 0.5))) +
          ((rx * rx) * ((y - 1) * (y - 1))) -
           (rx * rx * ry * ry);
     while (y >= 0)
    {
        plot(cx+x,cy+y);
        plot(cx-x,cy+y);
        plot(cx+x,cy-y);
        plot(cx-x,cy-y);
        if (p2 > 0)
        {
            y--;
            dy = dy - (2 * rx * rx);
            p2 = p2 + (rx * rx) - dy;
        }
        else
        {
            y--;
            x++;
            dx = dx + (2 * ry * ry);
            dy = dy - (2 * rx * rx);
            p2 = p2 + dx - dy + (rx * rx);
        }
    }
}
void figure_util(){
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(1.0);
    glColor3f(0,0,1);
    midpoint_ellipse_util(rx,ry,cx,cy);
    midpoint_ellipse_util(ry,rx,cx,cy);
    glutSwapBuffers();
}


int main(int argc, char *argv[]){
```

```
    cout<<"Draw Figure\n>>";
    cout<<"RX: ";
    cin >> rx;
    cout<<"RY: ";
    cin >> ry;
    cout<<"CX: ";
    cin >> cx;
    cout<<"CY: ";
    cin >> cy;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Assignment 6");
    glShadeModel(GLU_FLAT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 800.0, 0.0, 600.0);

    glutDisplayFunc(figure_util);

    glutMainLoop();
}
```


Assignment 6

2. Write a menu driven Program to implement set of basic Transformations on Polygon: Program should include: Translation, Rotation and Scaling.

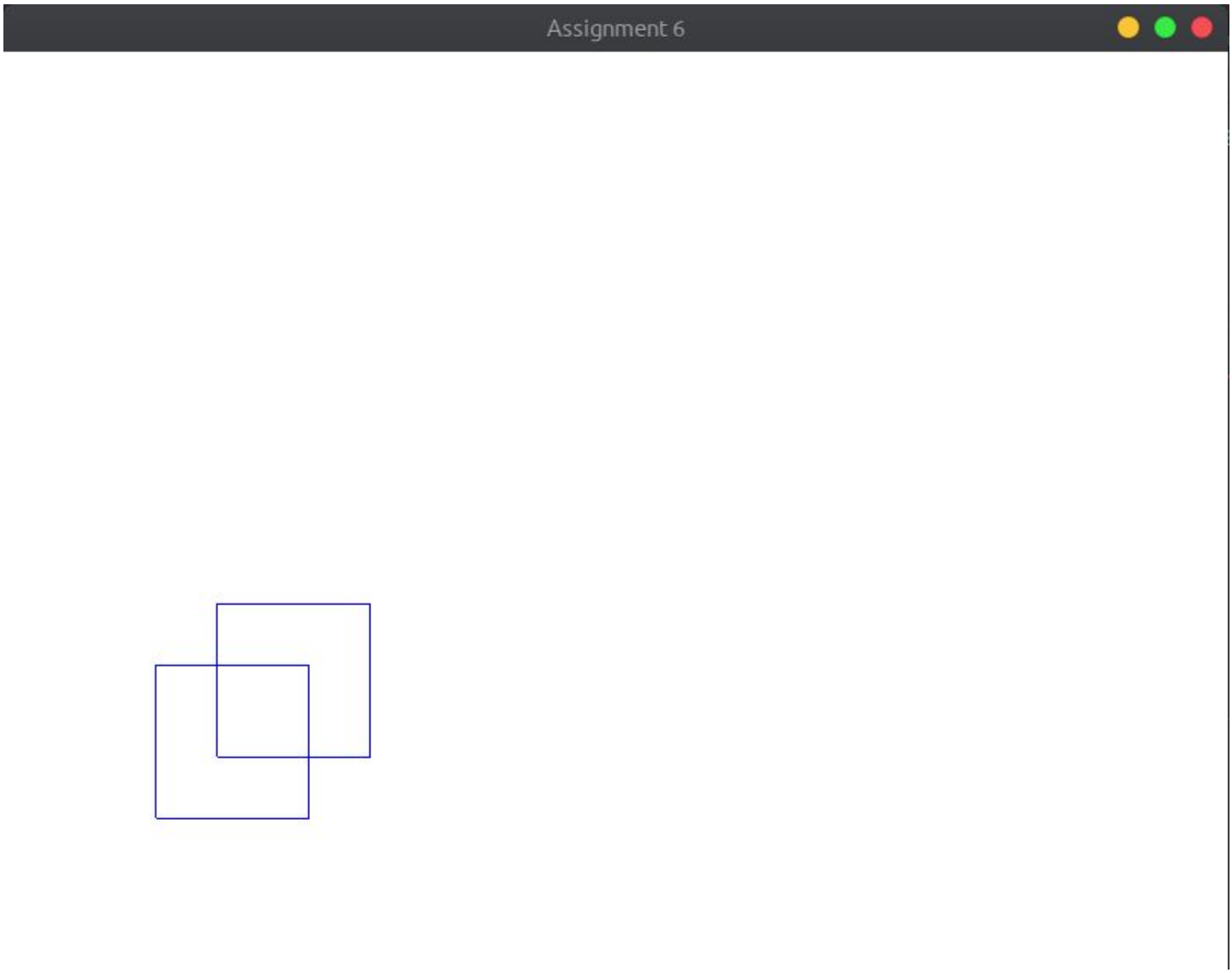
```
#include <bits/stdc++.h>
#include <GL/glut.h>

using namespace std;

// To Compile:-
// g++ -pthread 2.cpp -lglfw3 -lGLEW -lGLU -lGL -lXrandr -lXxf86vm -lXi -lXinerama
-lX11 -lrt -ldl -lglut

int n;
vector<vector<double>> points;
int choice;
double angle,tx,ty,s;



vector<vector<double>> matrix_mul(vector<vector<double>> &mat1,
vector<vector<double>> &mat2)
{
   int p = mat1.size();
   int q = mat1[0].size();
   int r = mat2[0].size();
   vector<vector<double>> mat3(p, vector<double>(q, 0));
   for (int i = 0; i < p; i++)
   {
       for (int k = 0; k < r; k++)
       {
           for (int j = 0; j < q; j++)
           {
               mat3[i][k] += mat1[i][j] * mat2[j][k];
           }
       }
   }
   return mat3;
}

vector<vector<double>> translateMath(vector<vector<double>> &points, int tX, int tY)
{
   vector<vector<double>> translateMatrix = {{1, 0, 0}, {0, 1, 0}, {(double)tX,
(double)tY, 1}};
   return matrix_mul(points, translateMatrix);
```

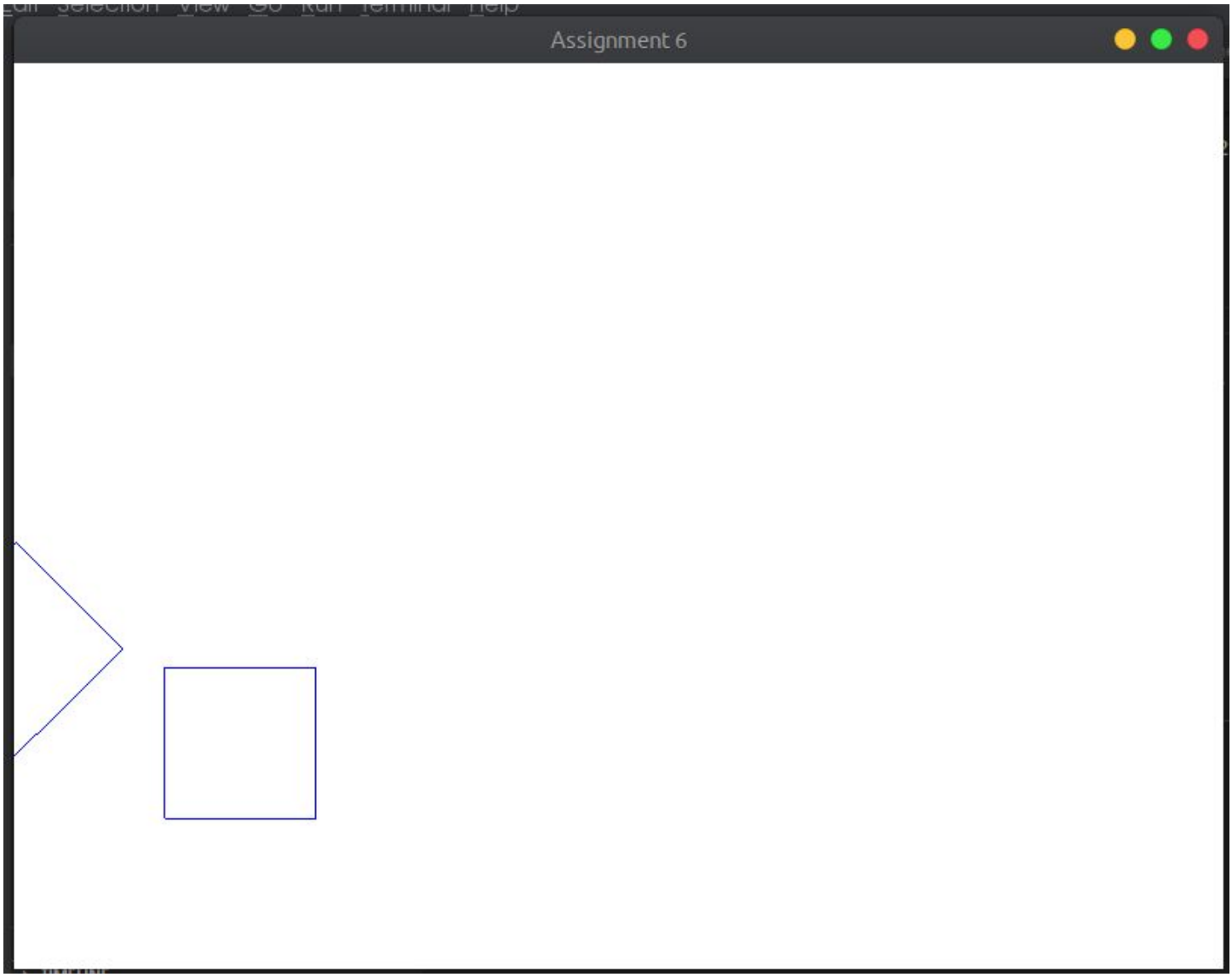
```
}


vector<vector<double>> rotateMath(vector<vector<double>> &points, double angle)
{
   vector<vector<double>> rotateMatrix = {{cos(angle),sin(angle), 0}, {-sin(angle),
cos(angle), 0}, {0, 0, 1}};
   return matrix_mul(points, rotateMatrix);
}


vector<vector<double>> scaleMath(vector<vector<double>> &points, double scale)
{
   vector<vector<double>> scaleMatrix = {{scale,0, 0}, {0, scale, 0}, {0, 0, 1}};
   return matrix_mul(points, scaleMatrix);
}

vector<vector<double>> getHomogeneousPoints(vector<vector<double>>& points){

   vector<vector<double>> homogeneousPoints(n, vector<double>(3, 1));
   for (int i = 0; i < n; i++)
       for (int j = 0; j < 2; j++)
           homogeneousPoints[i][j] = points[i][j];
   return homogeneousPoints;
}


void plotVector(vector<vector<double>>& points){

   glColor3f(0, 0, 1);
   for(int i = 0;i<n;i++){
       glBegin(GL_LINES);
       glVertex2f(points[i][0],points[i][1]);
       glVertex2f(points[(i+1)%n][0],points[(i+1)%n][1]);
       glEnd();
   }
}

void normal_util()
{
   plotVector(points);
}

void translate_util()
{
```

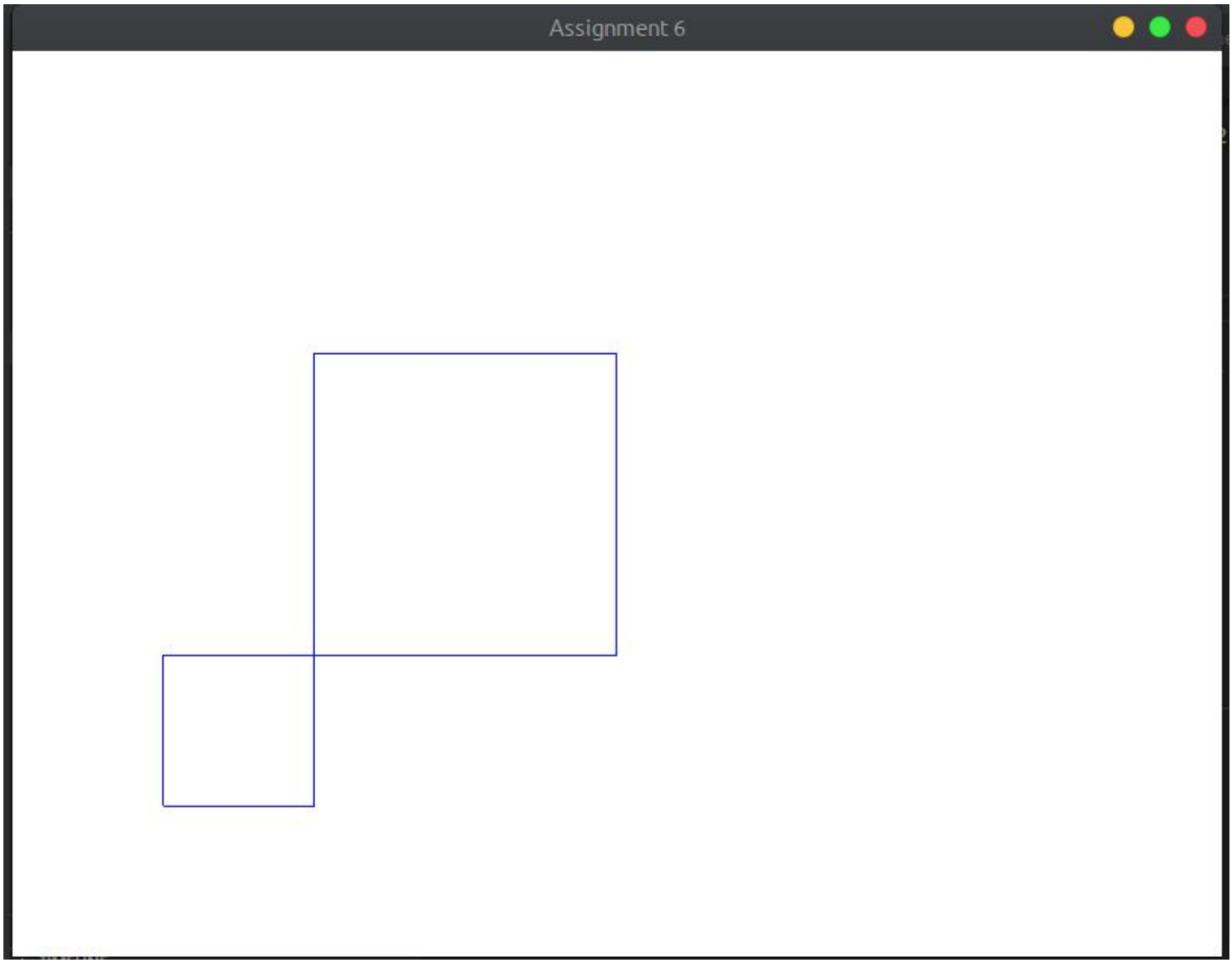
```
    vector<vector<double>> translatePoints = getHomogeneousPoints(points);
    vector<vector<double>> finalPoints = translateMath(translatePoints, tx, ty);
    plotVector(finalPoints);
}

void rotate_util()
{
    vector<vector<double>> rotatePoints = getHomogeneousPoints(points);
    vector<vector<double>> finalPoints = rotateMath(rotatePoints, angle);
    plotVector(finalPoints);
}

void scale_util()
{
    vector<vector<double>> scalePoints = getHomogeneousPoints(points);
    vector<vector<double>> finalPoints = scaleMath(scalePoints,s);
    plotVector(finalPoints);
}

void translate()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(1.0);
    glColor3f(0, 0, 1);
    normal_util();
    translate_util();
    glutSwapBuffers();
}

void rotate()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(1.0);
    glColor3f(0, 0, 1);
    normal_util();
    rotate_util();
    glutSwapBuffers();
}

void scale()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(1.0);
    glColor3f(0, 0, 1);
    normal_util();
```

```
    scale_util();
    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    cout << "Enter Number of Points: ";
    cin >> n;
    points = vector<vector<double>>(n, vector<double>(2, 0));
    for (int i = 0; i < n; i++)
    {
        cout << "Enter point[" << i << "].x: ";
        cin >> points[i][0];
        cout << "Enter point[" << i << "].y: ";
        cin >> points[i][1];
    }
    cout << "Enter Choice:\n1. Translate\n2. Rotate\n3. Scale\n>>";
    cin >> choice;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Assignment 6");
    glShadeModel(GLU_FLAT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 800.0, 0.0, 600.0);
    if (choice == 1)
    {
        cout<<"TX: ";
        cin >> tx;
        cout<<"TY: ";
        cin >> ty;
        glutDisplayFunc(translate);
    }
    else if (choice == 2)
    {
        cout<<"Angle: ";
        cin >> angle;
        glutDisplayFunc(rotate);
    }
    else if (choice == 3)
    {
        cout<<"Scale: ";
        cin >> s;
```

```
        glutDisplayFunc(scale);
    }
    else
    {
        cout << "Invalid Choice!" << endl;
    }

    glutMainLoop();
}

}
```




Assignment 6


Assignment 6

3.Write a menu driven program to implement set of Composite Transformations on Polygon:
Program should include: Translation, Rotation (about arbitrary point, arbitrary axis, and arbitrary plane), Scaling (fixed point), and Shearing (X & Y), Reflection (along X axis, along y axis, along the origin and along Y=X line).

```
#include <bits/stdc++.h>
#include <GL/glut.h>

using namespace std;

// To Compile:-
// g++ -pthread 3.cpp -lglfw3 -lGLEW -lGLU -lGL -lXrandr -lXxf86vm -lXi -lXinerama
-lX11 -lrt -ldl -lglut

int n;
vector<vector<double>> points;
int choice;
double angle, tx, ty, s, shear;
int isx;

void debug(){
   for(auto i: points){
       for(auto j: i) cout<<j<<" ";
       cout<<endl;
   }
}

vector<vector<double>> matrix_mul(vector<vector<double>> &mat1,
vector<vector<double>> &mat2)
{
   int p = mat1.size();
   int q = mat1[0].size();
   int r = mat2[0].size();
   vector<vector<double>> mat3(p, vector<double>(q, 0));
   for (int i = 0; i < p; i++)
   {
       for (int k = 0; k < r; k++)
       {
           for (int j = 0; j < q; j++)
           {
               mat3[i][k] += mat1[i][j] * mat2[j][k];
           }
       }
   }
```

```
    return mat3;
}

vector<vector<double>> translateMath(vector<vector<double>> &points, int tX, int tY)
{
    vector<vector<double>> translateMatrix = {{1, 0, 0}, {0, 1, 0}, {(double)tX,
(double)tY, 1}};
    return matrix_mul(points, translateMatrix);
}

vector<vector<double>> rotateMath(vector<vector<double>> &points, double angle)
{
    vector<vector<double>> rotateMatrix = {{cos(angle), sin(angle), 0}, {-sin(angle),
cos(angle), 0}, {0, 0, 1}};
    return matrix_mul(points, rotateMatrix);
}

vector<vector<double>> scaleMath(vector<vector<double>> &points, double scale)
{
    vector<vector<double>> scaleMatrix = {{scale, 0, 0}, {0, scale, 0}, {0, 0, 1}};
    return matrix_mul(points, scaleMatrix);
}

vector<vector<double>> shearMath(vector<vector<double>> &points, double shear, bool
isX)
{
    vector<vector<double>> scaleMatrix = {{1.0, isX == false ? shear : 0, 0}, {isX ==
true ? shear : 0, 1, 0}, {0, 0, 1}};
    return matrix_mul(points, scaleMatrix);
}

vector<vector<double>> getHomogeneousPoints(vector<vector<double>> &points)
{

    vector<vector<double>> homogeneousPoints(n, vector<double>(3, 1));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < 2; j++)
            homogeneousPoints[i][j] = points[i][j];
    return homogeneousPoints;
}

void plotVector(vector<vector<double>> &points)
{

    glColor3f(0, 0, 1);
```

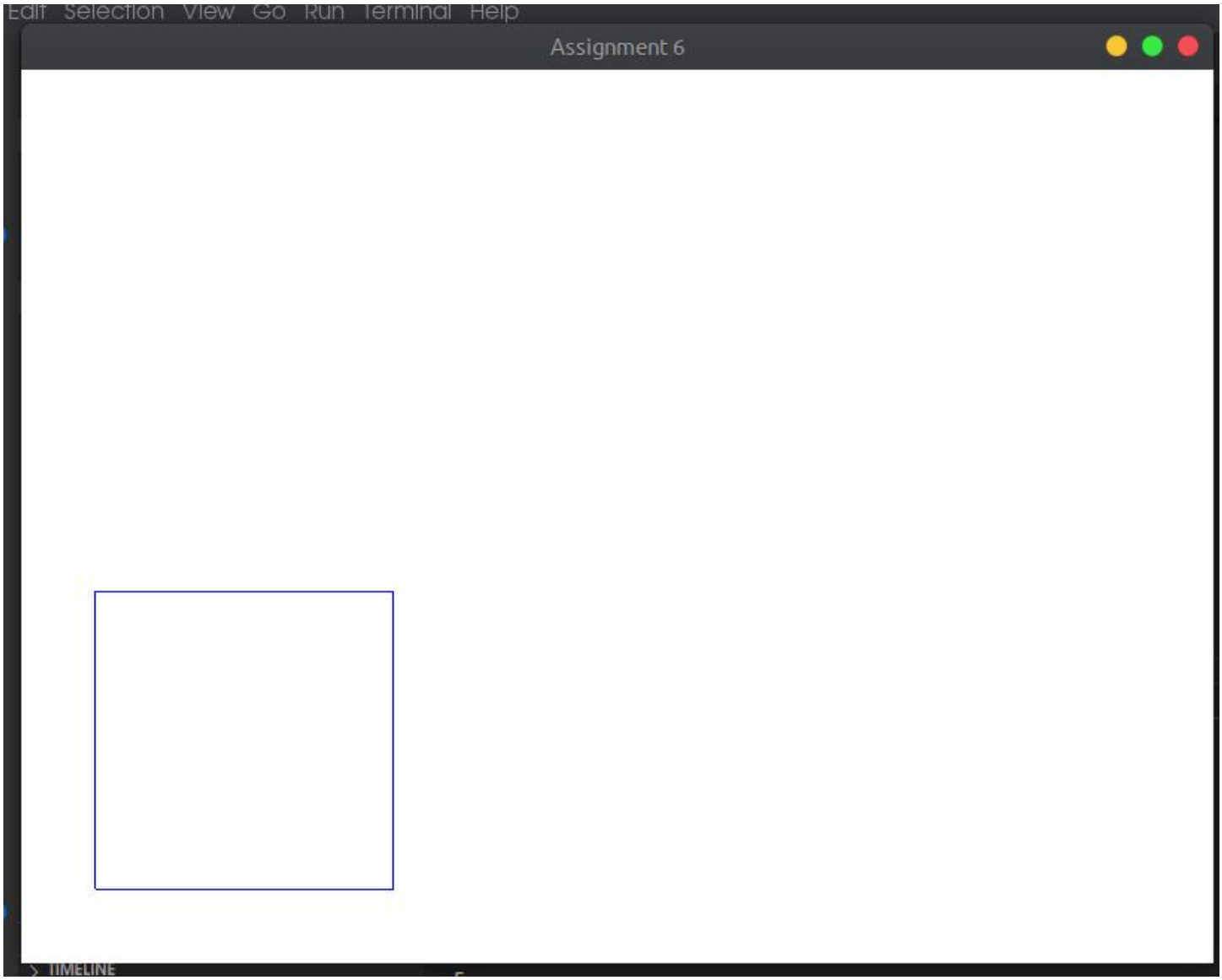
```
    for (int i = 0; i < n; i++)
    {
        glBegin(GL_LINES);
        glVertex2f(points[i][0], points[i][1]);
        glVertex2f(points[(i + 1) % n][0], points[(i + 1) % n][1]);
        glEnd();
    }
}

void normal_util()
{
    plotVector(points);
}

void translate_util()
{
    vector<vector<double>> translatePoints = getHomogeneousPoints(points);
    points = translateMath(translatePoints, tx, ty);
}

void rotate_util()
{
    vector<vector<double>> rotatePoints = getHomogeneousPoints(points);
    points = rotateMath(rotatePoints, angle);
}

void scale_util()
{
    vector<vector<double>> scalePoints = getHomogeneousPoints(points);
    points = scaleMath(scalePoints, s);
}

void shear_util()
{
    vector<vector<double>> scalePoints = getHomogeneousPoints(points);
    points = shearMath(scalePoints, shear, isx);
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(1.0);
    glColor3f(0, 0, 1);
    plotVector(points);
    glutSwapBuffers();
```

```
}


int main(int argc, char *argv[])
{
   cout << "Enter Number of Points: ";
   cin >> n;
   points = vector<vector<double>>(n, vector<double>(2, 0));
   for (int i = 0; i < n; i++)
   {
       cout << "Enter point[" << i << "].x: ";
       cin >> points[i][0];
       cout << "Enter point[" << i << "].y: ";
       cin >> points[i][1];
   }

   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize(800, 600);
   glutCreateWindow("Assignment 6");
   glShadeModel(GLU_FLAT);
   glClearColor(1.0, 1.0, 1.0, 0.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0.0, 800.0, 0.0, 600.0);
   while (true)
   {
       cout << "Enter Choice:\n1. Translate\n2. Rotate\n3. Scale\n4. Shear\n5.
Display\n>>";
       cin >> choice;
       cout<<choice<<endl;
       if (choice == 1)
       {
           cout << "TX: ";
           cin >> tx;
           cout << "TY: ";
           cin >> ty;
           translate_util();
       }
       else if (choice == 2)
       {
           cout << "Angle: ";
           cin >> angle;
           angle *= 180/(3.14159358979);
           rotate_util();
       }
```

```
        else if (choice == 3)
        {
            cout << "Scale: ";
            cin >> s;
            scale_util();
        }
        else if (choice == 4)
        {
            cout << "Shear: ";
            cin >> shear;
            cout << "1 if X Shear, 0 otherwise: ";
            cin >> isx;
            shear_util();
        }
        else if (choice == 5)
        {
            break;
        }
        else{
            cout << "Invalid Choice!" << endl;
        }
    }

    glutDisplayFunc(display);

    glutMainLoop();
}
```

```
krhero@hellblazer:/mnt/0FB812900FB81290/BTech/Assignments/3rd_Year/CG/Assignment6$ ./a.out
Enter Number of Points: 4
Enter point[0].x: 100
Enter point[0].y: 100
Enter point[1].x: 100
Enter point[1].y: 200
Enter point[2].x: 200
Enter point[2].y: 200
Enter point[3].x: 200
Enter point[3].y: 100
Enter Choice:
1. Translate
2. Rotate
3. Scale
4. Shear
5. Display
>>1
1
TX: -150
TY: -150
Enter Choice:
1. Translate
2. Rotate
3. Scale
4. Shear
5. Display
>>3
3
Scale: 2
Enter Choice:
1. Translate
2. Rotate
3. Scale
4. Shear
5. Display
>>1
1
TX: 150
```

```
TY: 150
Enter Choice:
1. Translate
2. Rotate
3. Scale
4. Shear
5. Display
>>5
5
```

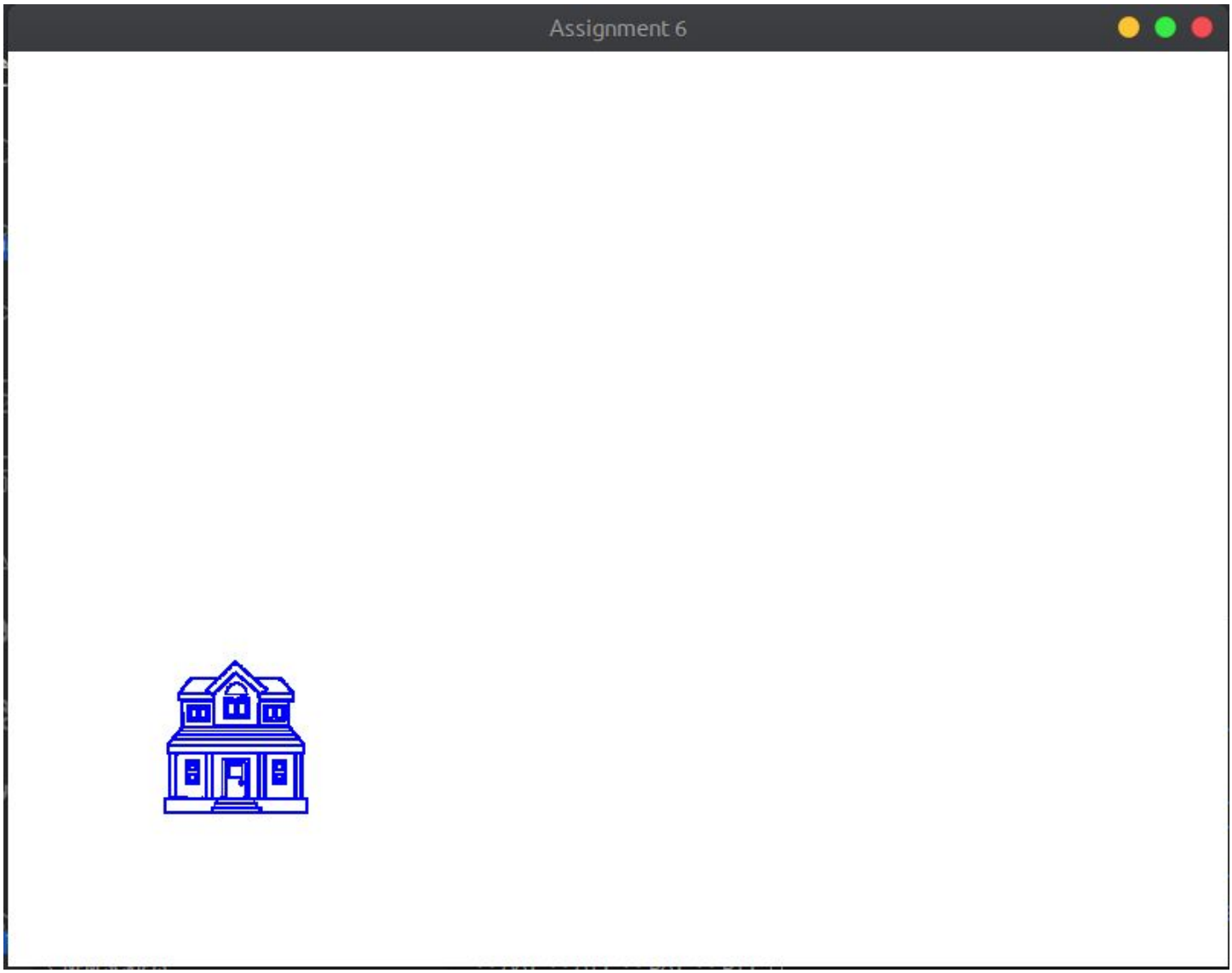

Edit Selection View Go Run Terminal Help
Assignment 6
TIMELINE

4. Write a program to draw the following structure:

```
#include <bits/stdc++.h>
#include <GL/glut.h>

using namespace std;

// To Compile:-
// g++ -pthread 4.cpp -lglfw3 -lGLEW -lGLU -lGL -lXrandr -lXxf86vm -lXi -lXinerama
-lX11 -lrt -ldl -lglut

int ax, ay, bx, by;

int scalex(int p, int m1, int m2)
{
   return m1 + (p * (m2 - m1)) / 330;
}

int scaley(int p, int m1, int m2)
{
   return m1 + (p * (m2 - m1)) / 242;
}

void plot(int x, int y)
{
   x = scalex(x, ax, bx);
   y = scaley(242 - y, ay, by);
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   glEnd();
}

void figure_util(int ax, int ay, int bx, int by);

void figure_display()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glPointSize(1.0);
   glColor3f(0, 0, 1);
   figure_util(ax, ay, bx, by);
   glutSwapBuffers();
}

int main(int argc, char *argv[])
{
   freopen("1.txt", "r", stdin);
```

```
    cout << "Draw Figure\n>>";
    cout << "AX: ";
    cin >> ax;
    cout << ">>AY: ";
    cin >> ay;
    cout << ">>BX: ";
    cin >> bx;
    cout << ">>BY: ";
    cin >> by;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Assignment 6");
    glShadeModel(GLU_FLAT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 800.0, 0.0, 600.0);

    glutDisplayFunc(figure_display);

    glutMainLoop();
}
```

Assignment 6


Assignment 6

6. Write a program to continuously rotate an object about a pivot point.

```
#include <bits/stdc++.h>
#include <GL/glut.h>
#include <chrono>
#include <thread>

using namespace std;

// To Compile:-
// g++ -pthread 5.cpp -lglfw3 -lGLEW -lGLU -lGL -lXrandr -lXxf86vm -lXi -lXinerama
-lX11 -lrt -ldl -lglut

int n;
vector<vector<double>> points;
double cx = 0,cy = 0;
double cosVal = cos(0.01256637061);
double sinVal = sin(0.01256637061);

void debug(){
   for(auto i: points){
       for(auto j: i) cout<<j<<" ";
       cout<<endl;
   }
}

vector<vector<double>> matrix_mul(vector<vector<double>> &mat1,
vector<vector<double>> &mat2)
{
   int p = mat1.size();
   int q = mat1[0].size();
   int r = mat2[0].size();
   vector<vector<double>> mat3(p, vector<double>(q, 0));
   for (int i = 0; i < p; i++)
   {
       for (int k = 0; k < r; k++)
       {
           for (int j = 0; j < q; j++)
           {
               mat3[i][k] += mat1[i][j] * mat2[j][k];
           }
       }
   }
   return mat3;
}
```

```
vector<vector<double>> translateMath(vector<vector<double>> &points,double tx,double
ty)
{
   vector<vector<double>> translateMatrix = {{1, 0, 0}, {0, 1, 0}, {(double)tx,
(double)tx, 1}};
   return matrix_mul(points, translateMatrix);
}

vector<vector<double>> rotateMath(vector<vector<double>> &points)
{
   vector<vector<double>> rotateMatrix = {{cosVal, sinVal, 0}, {-sinVal, cosVal, 0},
{0, 0, 1}};
   return matrix_mul(points, rotateMatrix);
}

vector<vector<double>> getHomogeneousPoints(vector<vector<double>> &points)
{

   vector<vector<double>> homogeneousPoints(n, vector<double>(3, 1));
   for (int i = 0; i < n; i++)
       for (int j = 0; j < 2; j++)
           homogeneousPoints[i][j] = points[i][j];
   return homogeneousPoints;
}

void plotVector(vector<vector<double>> &points)
{

   glColor3f(0, 0, 1);
   for (int i = 0; i < n; i++)
   {
       glBegin(GL_LINES);
       glVertex2f(points[i][0], points[i][1]);
       glVertex2f(points[(i + 1) % n][0], points[(i + 1) % n][1]);
       glEnd();
   }
}

void normal_util()
{
   plotVector(points);
}

void translate_util(double tx,double ty)
{
```

```
   vector<vector<double>> translatePoints = getHomogeneousPoints(points);
   points = translateMath(translatePoints,tx,ty);
}

void rotate_util()
{
   vector<vector<double>> rotatePoints = getHomogeneousPoints(points);
   points = rotateMath(rotatePoints);
}


void transform(vector<vector<double>>& points){
   translate_util(-cx,-cy);
   rotate_util();
   translate_util(cx,cy);
}

void display()
{

   points = getHomogeneousPoints(points);
   for(auto i: points){
       cx += i[0];
       cy += i[1];
   }
   cx /= n;
   cy /= n;
   while(true){
       glClear(GL_COLOR_BUFFER_BIT);
       glPointSize(1.0);
       glColor3f(0, 0, 1);
       plotVector(points);
       glutSwapBuffers();
       transform(points);
       this_thread::sleep_for(chrono::milliseconds(60) );
       glFlush();
   }
}


int main(int argc, char *argv[])
{
   cout << "Enter Number of Points: ";
   cin >> n;
   points = vector<vector<double>>(n, vector<double>(2, 0));
```

```
    for (int i = 0; i < n; i++)
    {
        cout << "Enter point[" << i << "].x: ";
        cin >> points[i][0];
        cout << "Enter point[" << i << "].y: ";
        cin >> points[i][1];
    }

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Assignment 6");
    glShadeModel(GLU_FLAT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 800.0, 0.0, 600.0);

    glutDisplayFunc(display);
    glutMainLoop();
}
```