COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

UCS505 COMPUTER GRAPHICS

LAB ASSIGNMENTS

SUBMITTED TO: Mr. X

SUBMITTED BY: -

Name:
Roll Number:
Batch: COE

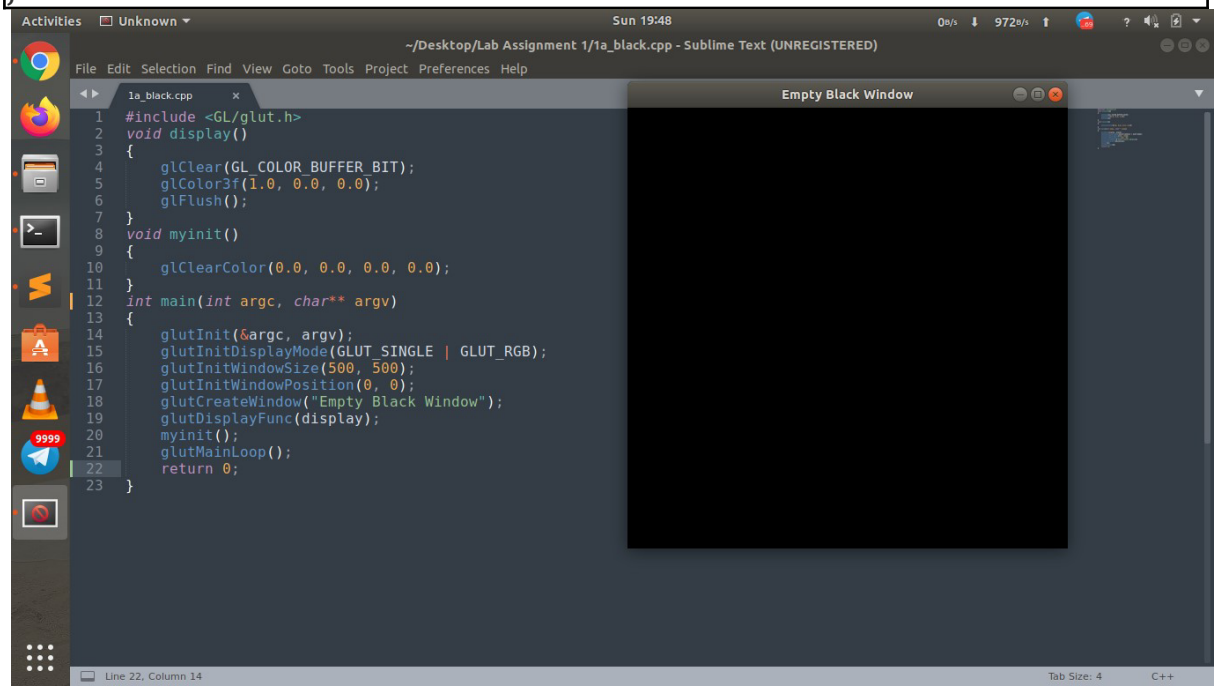
6th June 2021

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY,
PATIALA, PUNJAB

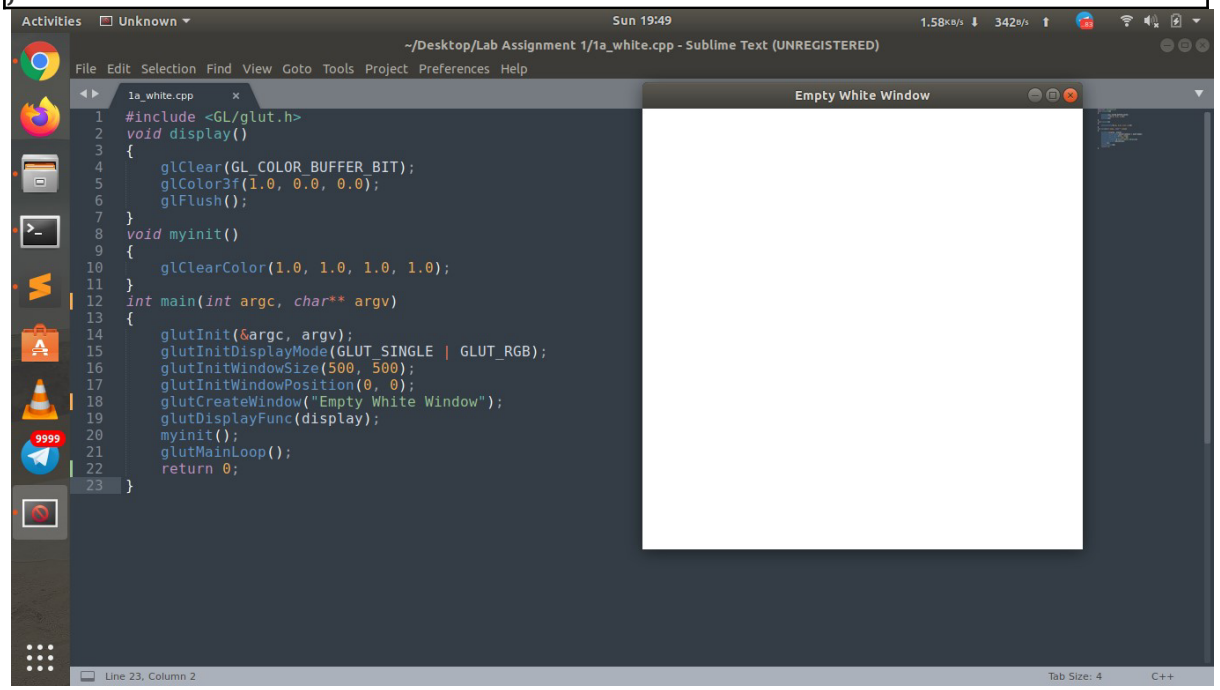
Table of Contents

Q1: a) Write a program to Create empty window (Black, White and different Colors)

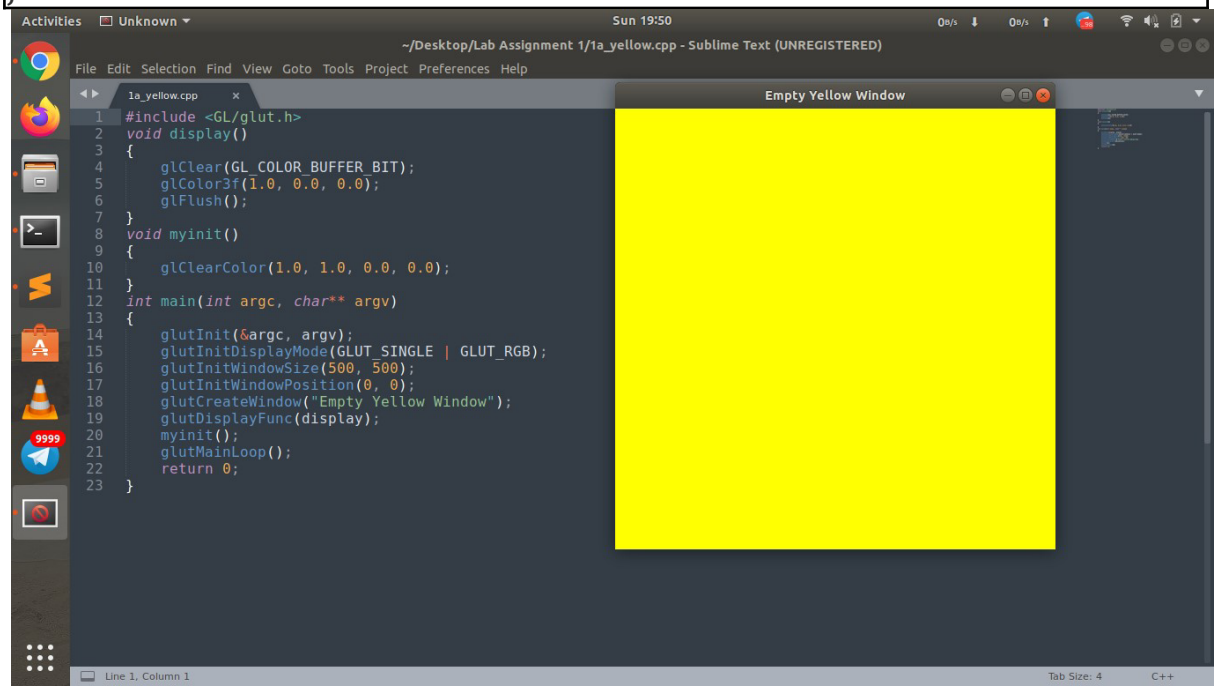
```
#include <GL/glut.h>
void display()
{ glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 0.0, 0.0);
   glFlush();
}
void myinit()
{ glClearColor(0.0, 0.0, 0.0,
   0.0);
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(0, 0);
   glutCreateWindow("Empty Black Window");
   glutDisplayFunc(display);
   myinit();
   glutMainLoop();
   return 0;
}
```



```
#include <GL/glut.h>
void display()
{ glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 0.0, 0.0);
   glFlush();
}
void myinit()
{ glClearColor(1.0, 1.0, 1.0,
   1.0);
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500,                500);
   glutInitWindowPosition(0,                 0);
   glutCreateWindow("Empty             White
   Window");
   glutDisplayFunc(display);
   myinit();
   glutMainLoop();
   return 0;
}
```

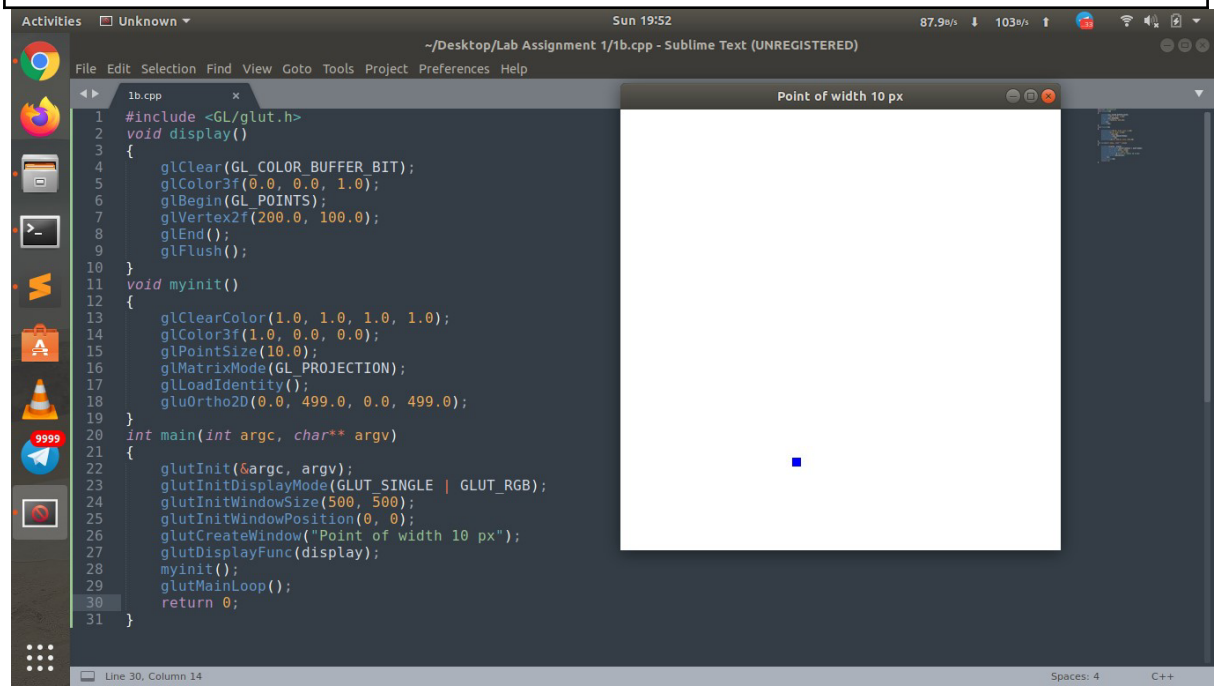


```
#include <GL/glut.h>
void display()
{ glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 0.0, 0.0);
   glFlush();
}
void myinit()
{ glClearColor(1.0, 1.0, 0.0,
   0.0);
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500,                500);
   glutInitWindowPosition(0,                 0);
   glutCreateWindow("Empty            Yellow
   Window");
   glutDisplayFunc(display);
   myinit();
   glutMainLoop();
   return 0;
}
```



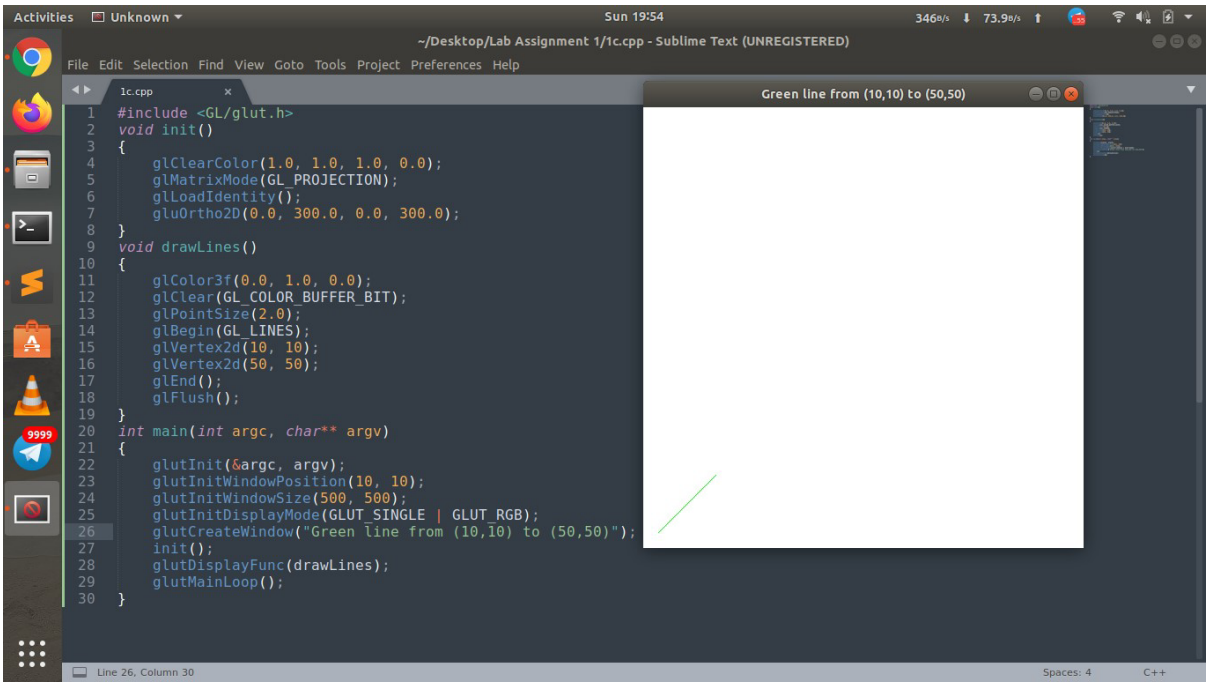
b) Write a program to Draw a point of width 10 pixel

```c
#include <GL/glut.h>
void display()
{ glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.0, 0.0, 1.0);
   glBegin(GL_POINTS);
   glVertex2f(200.0, 100.0);
   glEnd();
   glFlush();
}
void myinit()
{ glClearColor(1.0, 1.0, 1.0,
   1.0); glColor3f(1.0, 0.0, 0.0);
   glPointSize(10.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(0, 0);
   glutCreateWindow("Point of width 10 px");
   glutDisplayFunc(display);
   myinit();
   glutMainLoop();
   return 0;
}
```



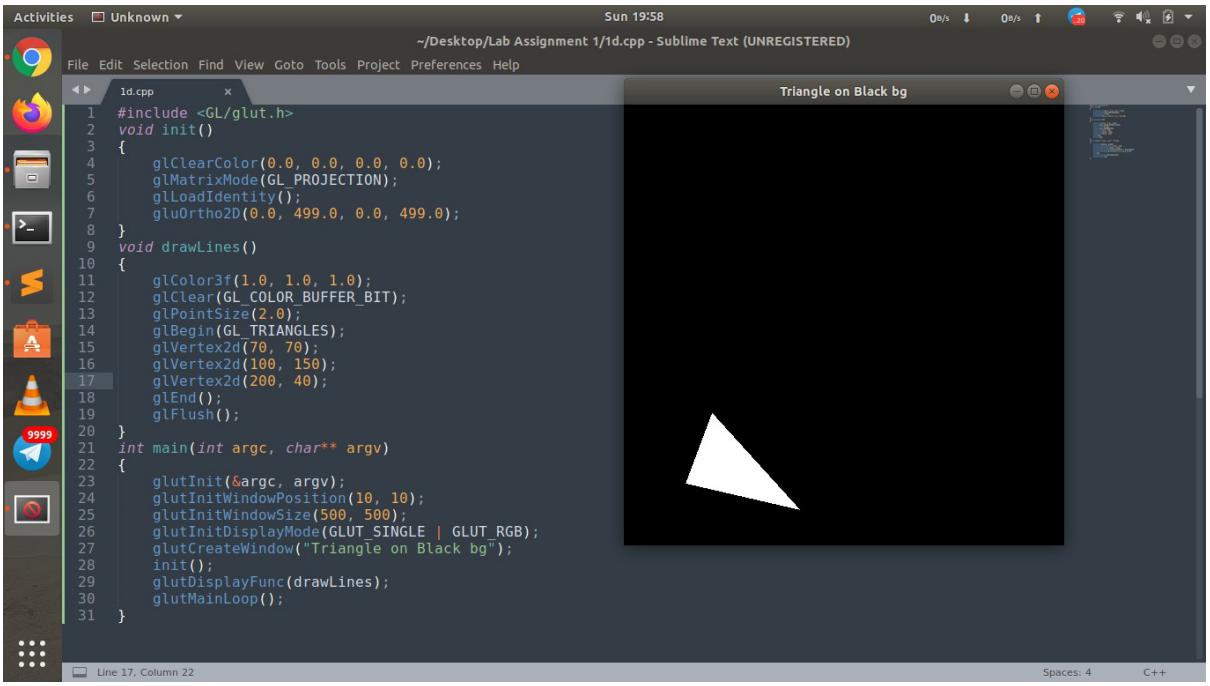
c) Write a program to Draw a green color line from (10,10) to (50,50)

```
#include <GL/glut.h>
void init()
{ glClearColor(1.0, 1.0, 1.0,
   0.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 300.0, 0.0, 300.0);
}
void drawLines()
{ glColor3f(0.0, 1.0,
   0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glPointSize(2.0);
   glBegin(GL_LINES);
   glVertex2d(10, 10);
   glVertex2d(50, 50);
   glEnd();
   glFlush();
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitWindowPosition(10, 10);
   glutInitWindowSize(500, 500);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutCreateWindow("Green line from (10,10) to (50,50)");
   init();
   glutDisplayFunc(drawLines);
   glutMainLoop();
}
```


Activities
Unknown
Sun 19:54
~/Desktop/Lab Assignment 1/1c.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1c.cpp
#include <GL/glut.h>
void init()
{
glClearColor(1.0, 1.0, 1.0, 0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 300.0, 0.0, 300.0);
}
void drawLines()
{
glColor3f(0.0, 1.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
glPointSize(2.0);
glBegin(GL_LINES);
glVertex2d(10, 10);
glVertex2d(50, 50);
glEnd();
glFlush();
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitWindowPosition(10, 10);
glutInitWindowSize(500, 500);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutCreateWindow("Green line from (10,10) to (50,50)");
init();
glutDisplayFunc(drawLines);
glutMainLoop();
}
Green line from (10,10) to (50,50)
Line 26, Column 30
Spaces: 4
C++

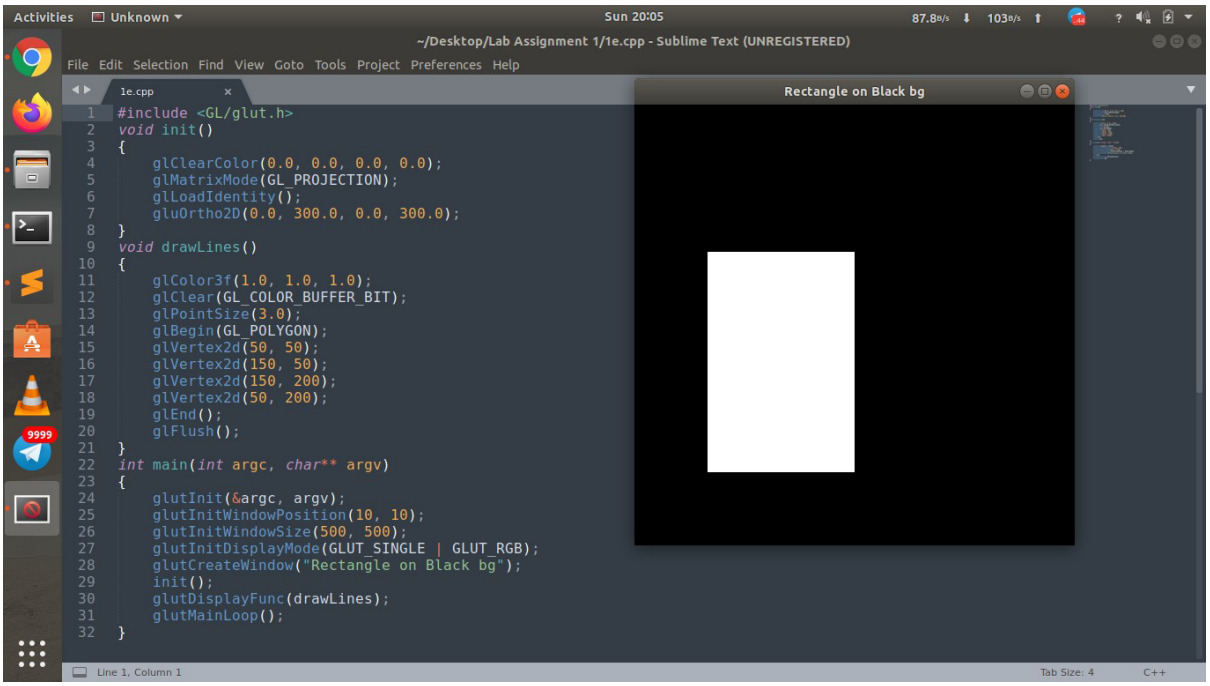
d) Write a program to Draw a triangle on black background

```
#include <GL/glut.h>
void init()
{ glClearColor(0.0, 0.0, 0.0,
   0.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}
void drawLines()
{ glColor3f(1.0, 1.0,
   1.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glPointSize(2.0);
   glBegin(GL_TRIANGLES);
   glVertex2d(70, 70);
   glVertex2d(100, 150);
   glVertex2d(200, 40);
   glEnd();
   glFlush();
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitWindowPosition(10, 10);
   glutInitWindowSize(500, 500);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutCreateWindow("Triangle on Black bg"); init();
   glutDisplayFunc(drawLines);
   glutMainLoop();
}
```


Activities
Unknown
Sun 19:58
~/Desktop/Lab Assignment 1/1d.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1d.cpp
#include <GL/glut.h>
void init()
{
glClearColor(0.0, 0.0, 0.0, 0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}
void drawLines()
{
glColor3f(1.0, 1.0, 1.0);
glClear(GL_COLOR_BUFFER_BIT);
glPointSize(2.0);
glBegin(GL_TRIANGLES);
glVertex2d(70, 70);
glVertex2d(100, 150);
glVertex2d(200, 40);
glEnd();
glFlush();
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitWindowPosition(10, 10);
glutInitWindowSize(500, 500);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutCreateWindow("Triangle on Black bg");
init();
glutDisplayFunc(drawLines);
glutMainLoop();
}
Triangle on Black bg
Line 17, Column 22
Spaces: 4
C++

e) Write a program to Draw a rectangle on black background

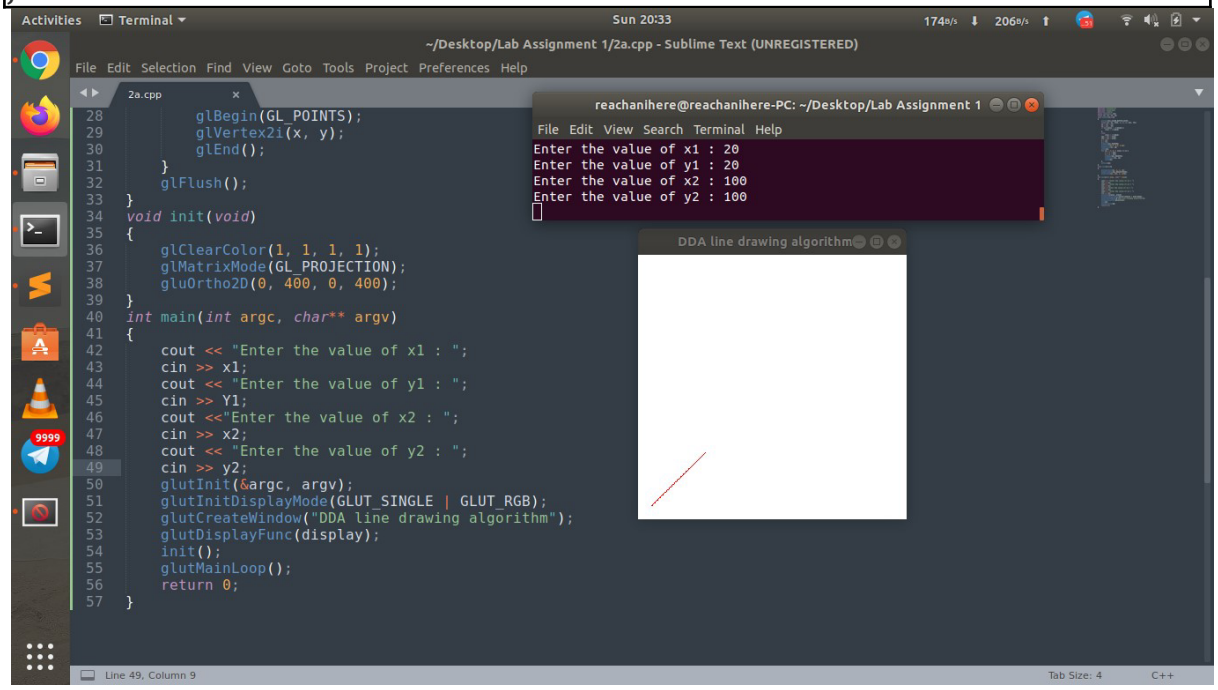
```
#include <GL/glut.h>
void init()
{ glClearColor(0.0, 0.0, 0.0,
   0.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 300.0, 0.0, 300.0);
}
void drawLines()
{ glColor3f(1.0, 1.0,
   1.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glPointSize(3.0);
   glBegin(GL_POLYGON);
   glVertex2d(50, 50);
   glVertex2d(150, 50);
   glVertex2d(150, 200);
   glVertex2d(50, 200);
   glEnd();
   glFlush();
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitWindowPosition(10, 10);
   glutInitWindowSize(500, 500);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutCreateWindow("Rectangle on Black bg");
   init();
   glutDisplayFunc(drawLines);
   glutMainLoop();
}
```


Activities
Unknown
Sun 20:05
~/Desktop/Lab Assignment 1/1e.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
1e.cpp
#include <GL/glut.h>
void init()
{
glClearColor(0.0, 0.0, 0.0, 0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 300.0, 0.0, 300.0);
}
void drawLines()
{
glColor3f(1.0, 1.0, 1.0);
glClear(GL_COLOR_BUFFER_BIT);
glPointSize(3.0);
glBegin(GL_POLYGON);
glVertex2d(50, 50);
glVertex2d(150, 50);
glVertex2d(150, 200);
glVertex2d(50, 200);
glEnd();
glFlush();
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitWindowPosition(10, 10);
glutInitWindowSize(500, 500);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutCreateWindow("Rectangle on Black bg");
init();
glutDisplayFunc(drawLines);
glutMainLoop();
}
Rectangle on Black bg
Line 1, Column 1
Tab Size: 4
C++

Q2: a) Write a program to draw a line using DDA algorithm

```
#include <GL/glut.h>
#include <iostream>
#include <cmath>
using namespace
std; float x1, x2, Y1,
y2; void display(void)
{ glClear(GL_COLOR_BUFFER_BIT);
   float dy, dx, step, x, y, k, Xin, Yin;
   dx = x2 - x1;
   dy = y2 - Y1;
   if (abs(dx) > abs(dy))
      { step = abs(dx);
   }
   else step =
      abs(dy);
   Xin = dx / step;
   Yin = dy / step;
   x = x1;
   y = Y1;
   glBegin(GL_POINTS);
   glColor3f(1.0, 0.0, 0.0);
   glVertex2i(x, y);
   glEnd();
   for (k = 1; k <= step; k++)
      { x = x + Xin;
      y = y + Yin;
      glBegin(GL_POINTS);
      glVertex2i(x, y);
      glEnd();
   }
   glFlush();
}
void init(void)
{ glClearColor(1, 1, 1, 1);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 400, 0, 400);
}
int main(int argc, char** argv)
{ cout << "Enter the value of x1 :
   "; cin >> x1;
   cout << "Enter the value of y1 : ";
   cin >> Y1;
   cout <<"Enter the value of x2 : ";
   cin >> x2;
   cout << "Enter the value of y2 : ";
```

```
    cin >> y2;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("DDA line drawing algorithm");
    glutDisplayFunc(display);
    init();
    glutMainLoop();
    return 0;
}
```

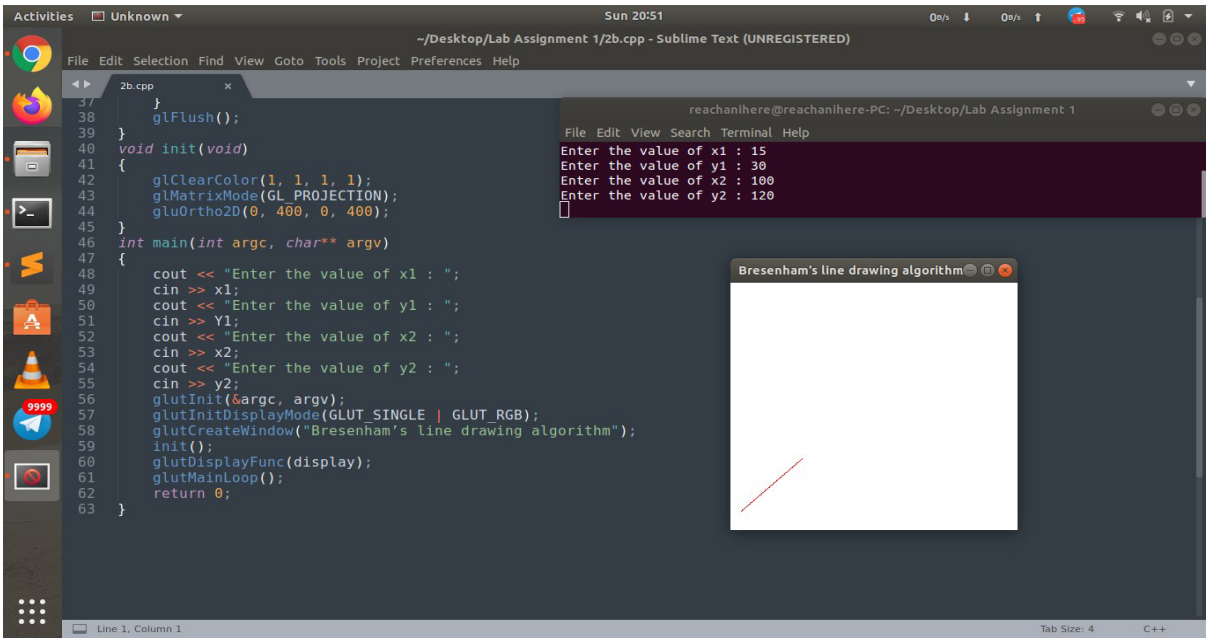


b) Write a program to draw a line using Bresenham's line algorithm

```
#include <GL/glut.h>
#include <iostream>
#include <cmath>
using namespace
std;

float x1, x2, Y1, y2;
void display(void)
{
    int x, y;
    x = x1;
    y = Y1;
    int dx, dy, pk, k, y_inc;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 0, 0);
    glBegin(GL_POINTS);
```

```
glVertex2i(x, y);
glEnd();
```

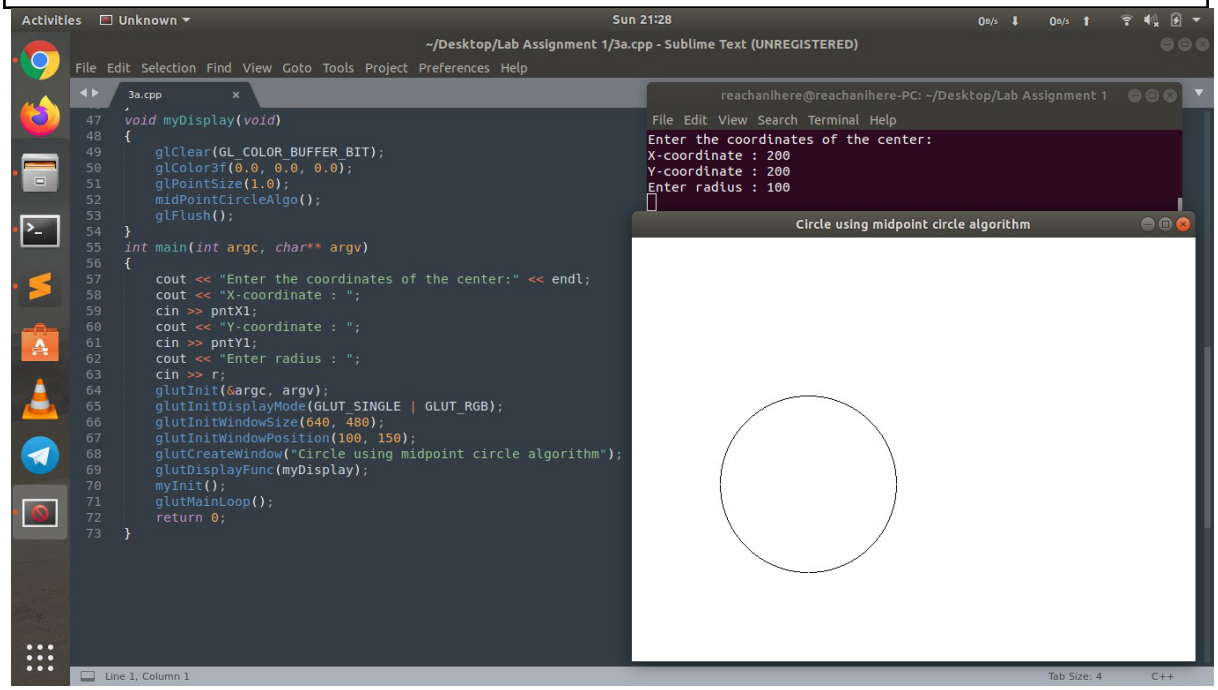
```
    dx = x2 - x1; dy
    = y2 - Y1; pk =
    2 * dy - dx; if
    (dx >= 0) y_inc
    = 1; else y_inc
    = -1;
    for (k = 0; k < abs(dx); k++)
        { if (pk < 0) {
            pk = pk + 2 * dy;
        }
        else {
            pk = pk + 2 * dy - 2 * dx;
            y = y + y_inc;
        } x++;
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
    }
    glFlush();
}
void init(void)
{ glClearColor(1, 1, 1, 1);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 400, 0, 400);
}
int main(int argc, char** argv)
{ cout << "Enter the value of x1 :
    "; cin >> x1;
    cout << "Enter the value of y1 : ";
    cin >> Y1;
    cout << "Enter the value of x2 : ";
    cin >> x2;
    cout << "Enter the value of y2 : ";
    cin >> y2;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("Bresenham's line drawing algorithm");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```


Activities
Unknown
Sun 20:51
~/Desktop/Lab Assignment 1/2b.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
2b.cpp
}
glFlush();
}
void init(void)
{
glClearColor(1, 1, 1, 1);
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0, 400, 0, 400);
}
int main(int argc, char** argv)
{
cout << "Enter the value of x1 : ";
cin >> x1;
cout << "Enter the value of y1 : ";
cin >> Y1;
cout << "Enter the value of x2 : ";
cin >> x2;
cout << "Enter the value of y2 : ";
cin >> y2;
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutCreateWindow("Bresenham's line drawing algorithm");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
reachanihere@reachanihere-PC: ~/Desktop/Lab Assignment 1
File Edit View Search Terminal Help
Enter the value of x1 : 15
Enter the value of y1 : 30
Enter the value of x2 : 100
Enter the value of y2 : 120
Bresenham's line drawing algorithm
Line 1, Column 1
Tab Size: 4
C++

Q3: a) Write a program to Draw a circle using Midpoint circle algorithm

```
#include <GL/glut.h>
#include <iostream>
using namespace std;

int pntX1, pntY1, r;
void plot(int x, int y)
{ glBegin(GL_POINTS);
   glVertex2i(x + pntX1, y +
   pntY1); glEnd();
}
void myInit(void)
{ glClearColor(1.0, 1.0, 1.0,
   0.0); glColor3f(0.0f, 0.0f,
   0.0f); glPointSize(4.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void midPointCircleAlgo()
{ int x = 0; int y = r; float
   decision = 5 / 4 - r;
   plot(x, y);
   while (y > x) { if
      (decision < 0)
      { x++;
         decision += 2 * x + 1;
      }
      else
      {
         y--;
         x++;
         decision += 2 * (x - y) + 1;
      }
      plot(x, y);
      plot(x, -y);
      plot(-x, y);
      plot(-x, -y);
      plot(y, x); plot(-
      y, x); plot(y, -x);
      plot(-y, -x);
   }
}
void myDisplay(void)
{
```

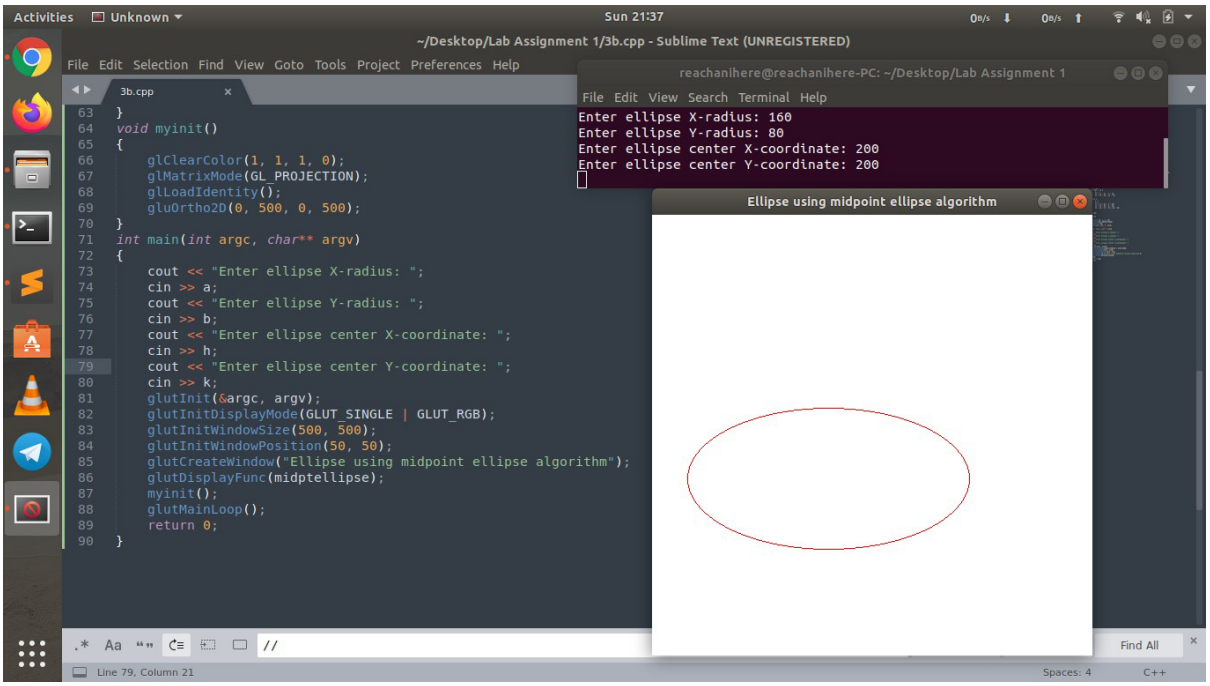
```
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glPointSize(1.0);
    midPointCircleAlgo();
    glFlush();
}
int main(int argc, char** argv)
{ cout << "Enter the coordinates of the center:" <<
    endl; cout << "X-coordinate : "; cin >> pntX1; cout
    << "Y-coordinate : "; cin >> pntY1;
    cout << "Enter radius : ";
    cin >> r;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 150);
    glutCreateWindow("Circle using midpoint circle
    algorithm"); glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 0;
}
```



b) Write a program to Draw an ellipse using Midpoint ellipse algorithm

```
#include <GL/glut.h>
#include <iostream>
using namespace std;

float a, b, h, k;
void
midptellipse()
{ glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1, 0, 0);
   float fx, fy, d1, d2, x, y;
   x = 0;
   y = b;

   d1 = (b * b) - (a * a * b) + (0.25 * a * a);
   fx = 2 * b * b * x;
   fy = 2 * a * a * y;

   while (fx < fy)
      { glBegin(GL_POINTS);
      glVertex2i(x + h, y + k);
      glVertex2i(-x + h, y + k);
      glVertex2i(x + h, -y + k);
      glVertex2i(-x + h, -y + k);
      glEnd(); if
      (d1 < 0)
      { x++;
         fx = fx + (2 * b * b);
         d1 = d1 + fx + (b * b);
      } else
      { x++;
         y--;
         fx = fx + (2 * b * b);
         fy = fy - (2 * a * a);
         d1 = d1 + fx - fy + (b * b);
      }
   } d2 = ((b * b) * ((x + 0.5) * (x + 0.5))) + ((a * a) * ((y - 1) * (y - 1))) - (a * a * b *

   b);

   while (y >= 0)
      { glBegin(GL_POINTS);

      glVertex2i(x + h, y + k);
      glVertex2i(-x + h, y + k);
      glVertex2i(x + h, -y + k);
      glVertex2i(-x + h, -y + k);
      glEnd();
```

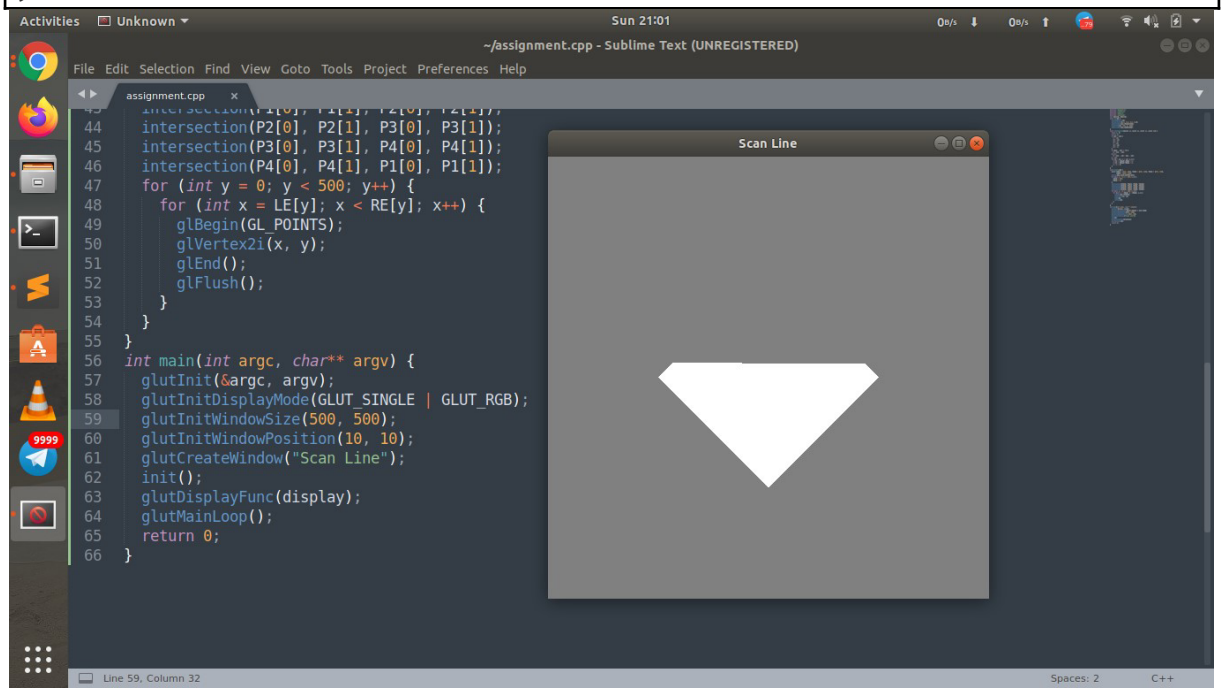
```
        if (d2 > 0) { y--;
            fy = fy - (2 * a * a);
            d2 = d2 + (a * a) - fy;
        }
        else
        {
            y--;
            x++;
            fx = fx + (2 * b * b);
            fy = fy - (2 * a * a);
            d2 = d2 + fx - fy + (a * a);
        }
    }
    glFlush();
}
void myinit()
{ glClearColor(1, 1, 1, 0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 500, 0, 500);
}
int main(int argc, char** argv)
{ cout << "Enter ellipse X-radius: ";
    cin >> a;
    cout << "Enter ellipse Y-radius: ";
    cin >> b;
    cout << "Enter ellipse center X-coordinate: ";
    cin >> h;
    cout << "Enter ellipse center Y-coordinate: ";
    cin >> k;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500); glutInitWindowPosition(50,
    50); glutCreateWindow("Ellipse using midpoint ellipse
    algorithm"); glutDisplayFunc(midptellipse); myinit();
    glutMainLoop(); return 0;
}
```

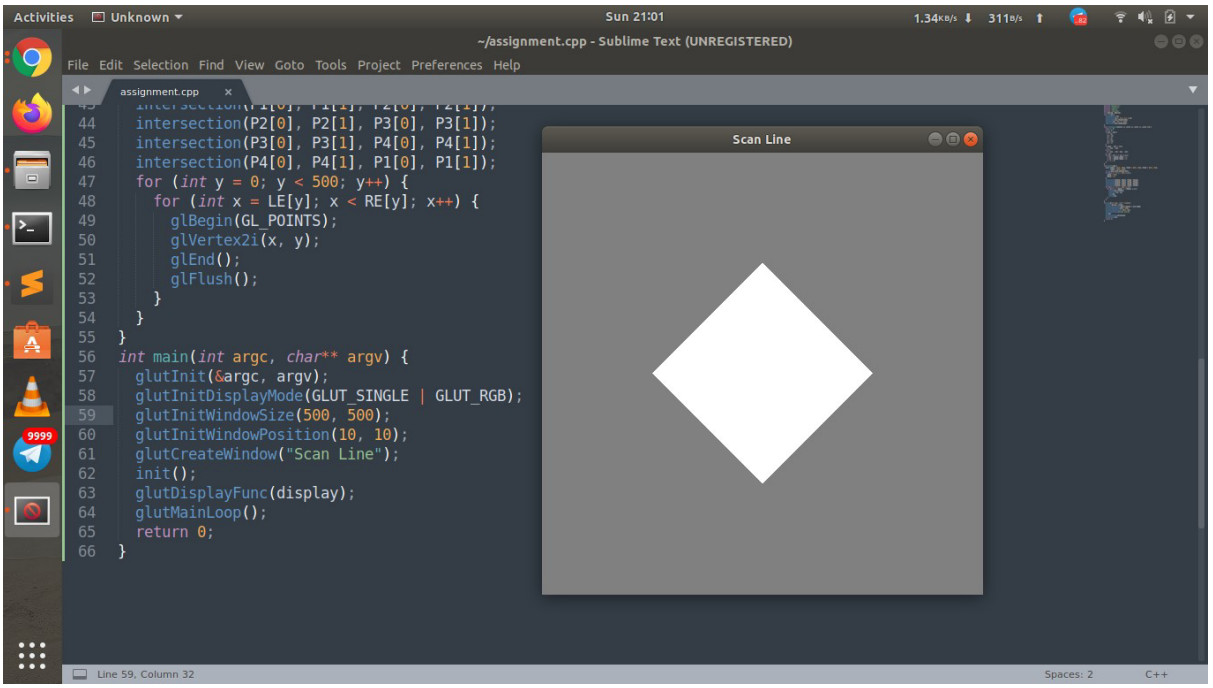

Activities
Unknown
Sun 21:37
~/Desktop/Lab Assignment 1/3b.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
3b.cpp
}
void myinit()
{
glClearColor(1, 1, 1, 0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, 500, 0, 500);
}
int main(int argc, char** argv)
{
cout << "Enter ellipse X-radius: ";
cin >> a;
cout << "Enter ellipse Y-radius: ";
cin >> b;
cout << "Enter ellipse center X-coordinate: ";
cin >> h;
cout << "Enter ellipse center Y-coordinate: ";
cin >> k;
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(50, 50);
glutCreateWindow("Ellipse using midpoint ellipse algorithm");
glutDisplayFunc(midptellipse);
myinit();
glutMainLoop();
return 0;
}
reachanihere@reachanihere-PC: ~/Desktop/Lab Assignment 1
File Edit View Search Terminal Help
Enter ellipse X-radius: 160
Enter ellipse Y-radius: 80
Enter ellipse center X-coordinate: 200
Enter ellipse center Y-coordinate: 200
Ellipse using midpoint ellipse algorithm
Find All
Line 79, Column 21
Spaces: 4
C++

Q4: Write a program to fill a polygon using scan line fill algorithm.

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
int LE[500], RE[500];
void init() {
  glLoadIdentity();
  glClearColor(0.5, 0.5, 0.5, 0.5);
  glMatrixMode(GL_PROJECTION)
  ; gluOrtho2D(0, 500, 0, 500);
  glMatrixMode(GL_MODELVIEW);
} void intersection(GLint x1, GLint y1, GLint x2, GLint
y2) {
  float x, M;
  int t, y;
  if (y1 > y2) {
    t = x1;
  x1 = x2;
  x2 = t; t
  = y1; y1
  = y2; y2
  = t; }
  if ((y2 - y1) == 0)
    { M = (x2 - x1);
  } else
    M = (x2 - x1) / (y2 - y1);
  x = x1;
  for (y = y1; y <= y2; y++) { if
    (x < LE[y]) LE[y] = x;
    if (x > RE[y]) RE[y] = x;
    x = x + M;
  }
} void
display() {
  GLint P1[2] = {125, 250}, P2[2] = {250, 125}, P3[2] = {375, 250},
        P4[2] = {250, 375}; glClear(GL_COLOR_BUFFER_BIT);
  for (int i = 0; i < 500; i++) {
    LE[i] = 500;
    RE[i] = 0;
  }
  intersection(P1[0], P1[1], P2[0], P2[1]);
  intersection(P2[0], P2[1], P3[0], P3[1]);
  intersection(P3[0], P3[1], P4[0], P4[1]);
  intersection(P4[0], P4[1], P1[0], P1[1]);
  for (int y = 0; y < 500; y++) {
    for (int x = LE[y]; x < RE[y]; x++) {
```

```
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
  }
 }
} int main(int argc, char**
argv) {
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 glutInitWindowSize(500, 500);
 glutInitWindowPosition(10, 10);
 glutCreateWindow("Scan Line");
 init();
 glutDisplayFunc(display);
 glutMainLoop();
 return 0;
}
```




Activities
Unknown
Sun 21:01
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
intersection(P2[0], P2[1], P3[0], P3[1]);
intersection(P3[0], P3[1], P4[0], P4[1]);
intersection(P4[0], P4[1], P1[0], P1[1]);
for (int y = 0; y < 500; y++) {
for (int x = LE[y]; x < RE[y]; x++) {
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
glFlush();
}
}
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(10, 10);
glutCreateWindow("Scan Line");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
Scan Line
Line 59, Column 32
Spaces: 2
C++

Q5: Write a program to fill a polygon using boundary fill and flood fill algorithm (4-connected and 8-connected) for various concave and convex polygons.

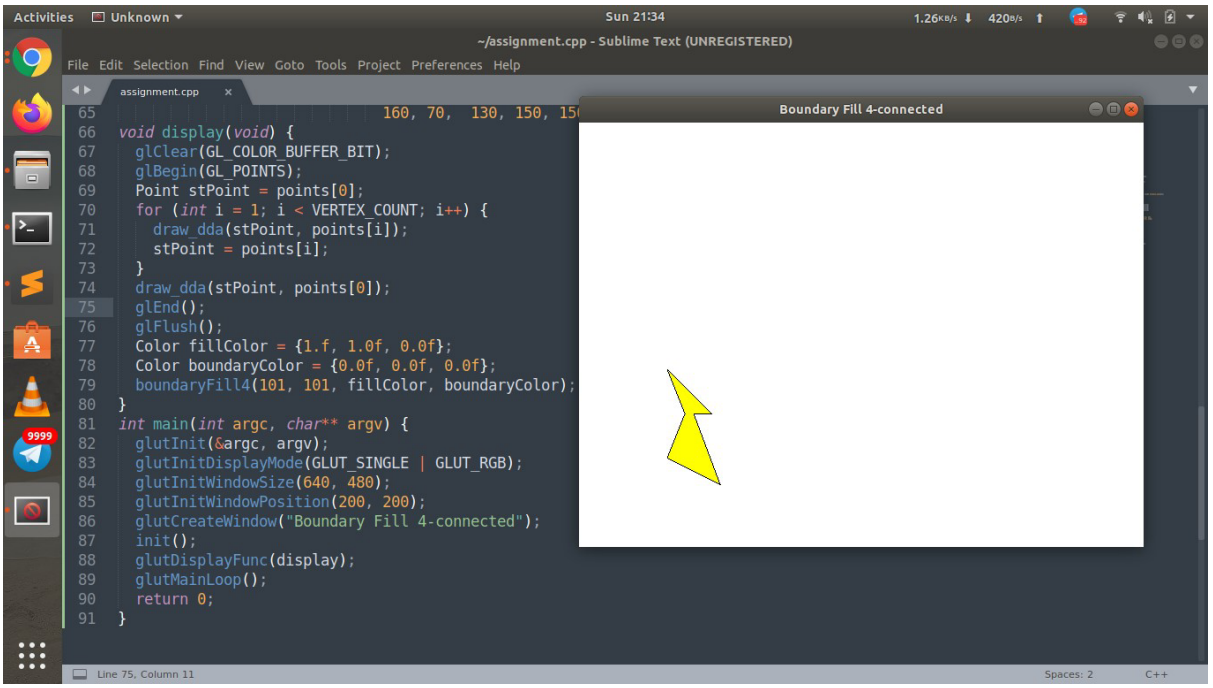
Boundary Fill

4 – connected concave

```
#include <cmath>
#include <GL/glut.h>
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
}; void
init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glPointSize(1.0f);
} void draw_dda(Point p1, Point
p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0; if
  (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else { step =
    abs(dy);
  }
  GLfloat xInc = dx / step;
  GLfloat yInc = dy / step; for
  (float i = 1; i <= step; i++) {
    glVertex2i(x1,
    y1); x1 += xInc;
    y1 += yInc;
  }
}
Color getPixelColor(GLint x, GLint y) {
  Color color;
  glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); return
  color;
```

```
}
```

```
void setPixelColor(GLint x, GLint y, Color color) {
  glColor3f(color.r, color.g, color.b);
  glBegin(GL_POINTS);
  glVertex2i(x, y);
  glEnd();
  glFlush();
} void boundaryFill4(int x, int y, Color fill_color, Color
boundary_color) { Color currentColor = getPixelColor(x, y); if
(currentColor.r != boundary_color.r && currentColor.g !=
boundary_color.g && currentColor.b != boundary_color.b) {
    setPixelColor(x, y, fill_color); boundaryFill4(x +
  1, y, fill_color, boundary_color); boundaryFill4(x,
  y + 1, fill_color, boundary_color);
  boundaryFill4(x - 1, y, fill_color, boundary_color);
  boundaryFill4(x, y - 1, fill_color, boundary_color);
   }
}
#define VERTEX_COUNT 6
Point points[VERTEX_COUNT] = {100, 200, 120, 150, 100, 100,
                      160, 70, 130, 150, 150, 150};
void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POINTS); Point stPoint =
  points[0]; for (int i = 1; i <
  VERTEX_COUNT; i++)
  { draw_dda(stPoint, points[i]); stPoint =
  points[i];
  }
  draw_dda(stPoint,
  points[0]); glEnd();
  glFlush();
  Color fillColor = {1.f, 1.0f, 0.0f}; Color
  boundaryColor = {0.0f, 0.0f, 0.0f};
  boundaryFill4(101, 101, fillColor, boundaryColor);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(200, 200);
  glutCreateWindow("Boundary Fill 4-connected");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

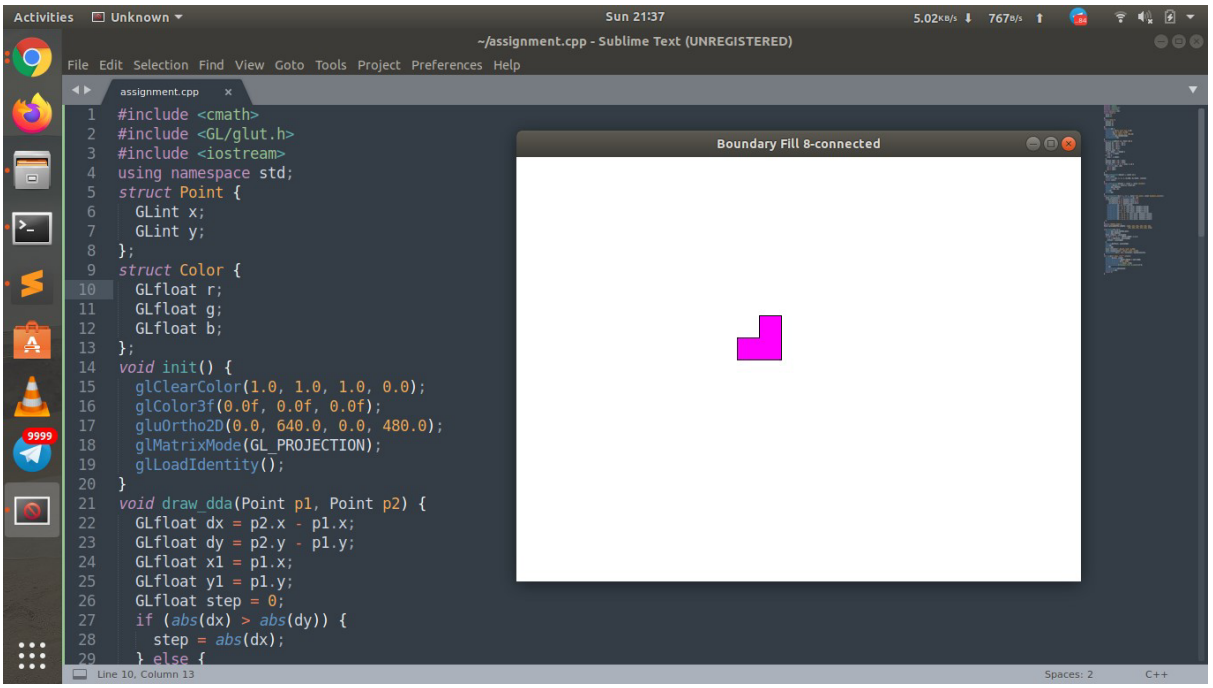

Activities
Unknown
Sun 21:34
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
Boundary Fill 4-connected
void display(void) {
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
Point stPoint = points[0];
for (int i = 1; i < VERTEX_COUNT; i++) {
draw_dda(stPoint, points[i]);
stPoint = points[i];
}
draw_dda(stPoint, points[0]);
glEnd();
glFlush();
Color fillColor = {1.f, 1.0f, 0.0f};
Color boundaryColor = {0.0f, 0.0f, 0.0f};
boundaryFill4(101, 101, fillColor, boundaryColor);
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(640, 480);
glutInitWindowPosition(200, 200);
glutCreateWindow("Boundary Fill 4-connected");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
Line 75, Column 11
Spaces: 2
C++

8 – connected concave

```
#include <cmath>
#include <GL/glut.h>
#include <iostream>
using namespace std;
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
}; void
init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
} void draw_dda(Point p1, Point
p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0; if
  (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else { step =
    abs(dy);
  }
  GLfloat xInc = dx / step;
  GLfloat yInc = dy / step; for
  (float i = 1; i <= step; i++) {
    glVertex2i(x1,
    y1); x1 += xInc;
    y1 += yInc;
  }
}
Color getPixelColor(GLint x, GLint y) {
  Color color;
  glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); return
  color;
} void setPixelColor(GLint x, GLint y, Color
color) {
  glColor3f(color.r, color.g, color.b);
```

```
glBegin(GL_POINTS);
glVertex2i(x, y);
```

```
  glEnd();
  glFlush();
} void boundaryFill8(int x, int y, Color fill_color, Color
boundary_color) { Color currentColor = getPixelColor(x, y); if
(currentColor.r != boundary_color.r && currentColor.g !=
boundary_color.g && currentColor.b != boundary_color.b) {
    setPixelColor(x, y, fill_color); boundaryFill8(x + 1, y,
  fill_color, boundary_color); boundaryFill8(x - 1, y,
  fill_color, boundary_color); boundaryFill8(x, y + 1,
  fill_color, boundary_color); boundaryFill8(x, y - 1,
  fill_color, boundary_color); boundaryFill8(x - 1, y - 1,
  fill_color, boundary_color); boundaryFill8(x - 1, y + 1,
  fill_color, boundary_color); boundaryFill8(x + 1, y - 1,
  fill_color, boundary_color); boundaryFill8(x + 1, y +
  1, fill_color, boundary_color); }
}
#define VERTEX_COUNT 6
Point points[VERTEX_COUNT] = {250, 250, 300, 250, 300, 300,
                      275, 300, 275, 275, 250, 275};
void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POINTS); Point stPoint =
  points[0]; for (int i = 1; i <
  VERTEX_COUNT; i++)
  { draw_dda(stPoint, points[i]); stPoint =
  points[i];
  }
  draw_dda(stPoint,
  points[0]); glEnd();
  glFlush();
  Color fillColor = {1.0f, 0.0f, 1.0f};
  Color boundaryColor = {0.0f, 0.0f,
  0.0f};
  boundaryFill8(251, 251, fillColor, boundaryColor);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(200, 200);
  glutCreateWindow("Boundary Fill 8-connected");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

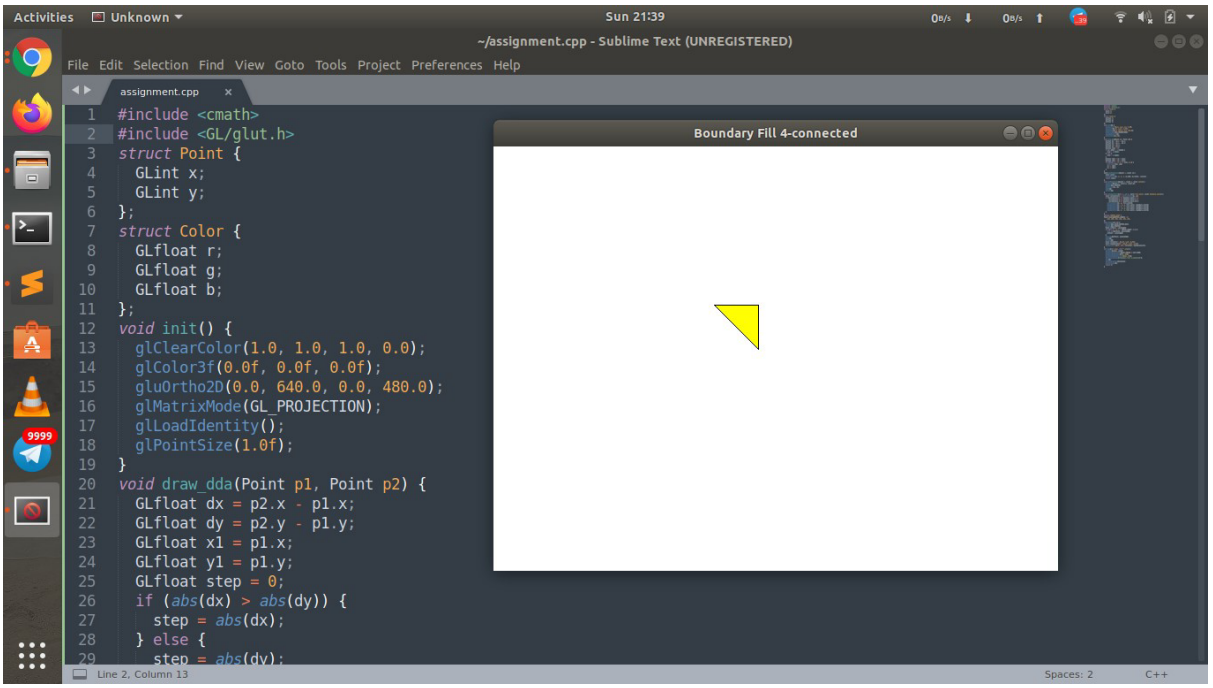

Activities
Unknown
Sun 21:37
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
#include <cmath>
#include <GL/glut.h>
#include <iostream>
using namespace std;
struct Point {
GLint x;
GLint y;
};
struct Color {
GLfloat r;
GLfloat g;
GLfloat b;
};
void init() {
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0f, 0.0f, 0.0f);
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
}
void draw_dda(Point p1, Point p2) {
GLfloat dx = p2.x - p1.x;
GLfloat dy = p2.y - p1.y;
GLfloat x1 = p1.x;
GLfloat y1 = p1.y;
GLfloat step = 0;
if (abs(dx) > abs(dy)) {
step = abs(dx);
} else {
Boundary Fill 8-connected
Line 10, Column 13
Spaces: 2
C++

4 – connected convex

```
#include <cmath>
#include <GL/glut.h>
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
}; void
init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glPointSize(1.0f);
} void draw_dda(Point p1, Point
p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0; if
  (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else { step =
    abs(dy);
  }
  GLfloat xInc = dx / step;
  GLfloat yInc = dy / step; for
  (float i = 1; i <= step; i++) {
    glVertex2i(x1,
    y1); x1 += xInc;
    y1 += yInc;
  }
}
Color getPixelColor(GLint x, GLint y) {
  Color color;
  glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); return
  color;
} void setPixelColor(GLint x, GLint y, Color
color) {
  glColor3f(color.r, color.g, color.b);
  glBegin(GL_POINTS);
```

```
glVertex2i(x, y);
glEnd();
```

```
  glFlush();
} void boundaryFill4(int x, int y, Color fill_color, Color
boundary_color) { Color currentColor = getPixelColor(x, y); if
(currentColor.r != boundary_color.r && currentColor.g !=
boundary_color.g && currentColor.b != boundary_color.b) {
    setPixelColor(x, y, fill_color); boundaryFill4(x +
  1, y, fill_color, boundary_color); boundaryFill4(x,
  y + 1, fill_color, boundary_color);
  boundaryFill4(x - 1, y, fill_color, boundary_color);
  boundaryFill4(x, y - 1, fill_color, boundary_color);
   }
}
#define VERTEX_COUNT 3
Point points[VERTEX_COUNT] = {
    250, 300, 300, 300, 300, 250,
}; void
display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POINTS); Point stPoint =
  points[0]; for (int i = 1; i <
  VERTEX_COUNT; i++)
  { draw_dda(stPoint, points[i]); stPoint =
  points[i];
  }
  draw_dda(stPoint,
  points[0]); glEnd();
  glFlush();
  Color fillColor = {1.0f, 1.0f, 0.0f};
  Color boundaryColor = {0.0f, 0.0f,
  0.0f};
  boundaryFill4(276, 275, fillColor, boundaryColor);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(200, 200);
  glutCreateWindow("Boundary Fill 4-connected");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```



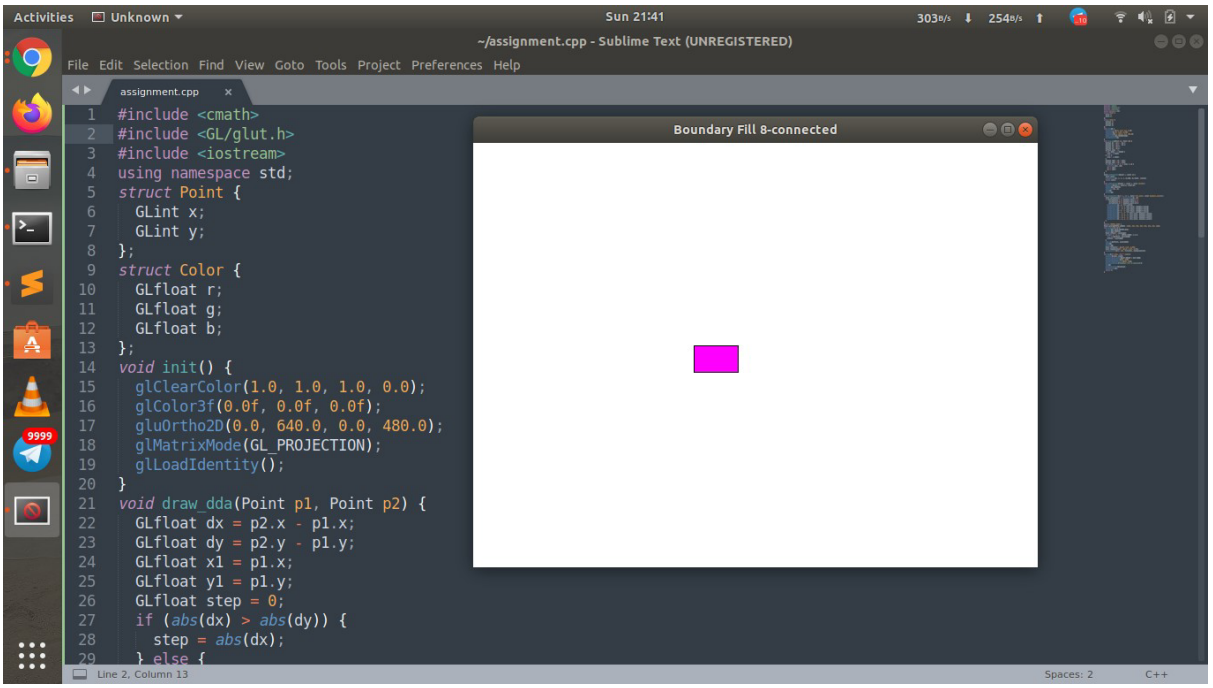

```
#include <cmath>
#include <GL/glut.h>
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
};
void init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glPointSize(1.0f);
}
void draw_dda(Point p1, Point p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0;
  if (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else {
    step = abs(dy);
```

8 – connected convex

```
#include <cmath>
#include <GL/glut.h>
#include <iostream>
using namespace std;
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
}; void
init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
} void draw_dda(Point p1, Point
p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0; if
  (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else { step =
    abs(dy);
  }
  GLfloat xInc = dx / step;
  GLfloat yInc = dy / step; for
  (float i = 1; i <= step; i++) {
    glVertex2i(x1,
    y1); x1 += xInc;
    y1 += yInc;
  }
}
Color getPixelColor(GLint x, GLint y) {
  Color color;
  glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); return
  color;
} void setPixelColor(GLint x, GLint y, Color
color) {
  glColor3f(color.r, color.g, color.b);
```

```
glBegin(GL_POINTS);
glVertex2i(x, y);
```

```
  glEnd();
  glFlush();
} void boundaryFill8(int x, int y, Color fill_color, Color
boundary_color) { Color currentColor = getPixelColor(x, y); if
(currentColor.r != boundary_color.r && currentColor.g !=
boundary_color.g && currentColor.b != boundary_color.b) {
    setPixelColor(x, y, fill_color); boundaryFill8(x + 1, y,
  fill_color, boundary_color); boundaryFill8(x - 1, y,
  fill_color, boundary_color); boundaryFill8(x, y + 1,
  fill_color, boundary_color); boundaryFill8(x, y - 1,
  fill_color, boundary_color); boundaryFill8(x - 1, y - 1,
  fill_color, boundary_color); boundaryFill8(x - 1, y + 1,
  fill_color, boundary_color); boundaryFill8(x + 1, y - 1,
  fill_color, boundary_color); boundaryFill8(x + 1, y +
  1, fill_color, boundary_color); }
}
#define VERTEX_COUNT 4
Point points[VERTEX_COUNT] = {250, 250, 300, 250, 300, 220, 250, 220};
void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POINTS); Point stPoint =
  points[0]; for (int i = 1; i <
  VERTEX_COUNT; i++)
  { draw_dda(stPoint, points[i]); stPoint =
  points[i];
  }
  draw_dda(stPoint,
  points[0]); glEnd();
  glFlush();
  Color fillColor = {1.0f, 0.0f, 1.0f};
  Color boundaryColor = {0.0f, 0.0f,
  0.0f};
  boundaryFill8(275, 235, fillColor, boundaryColor);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(200, 200);
  glutCreateWindow("Boundary Fill 8-connected");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

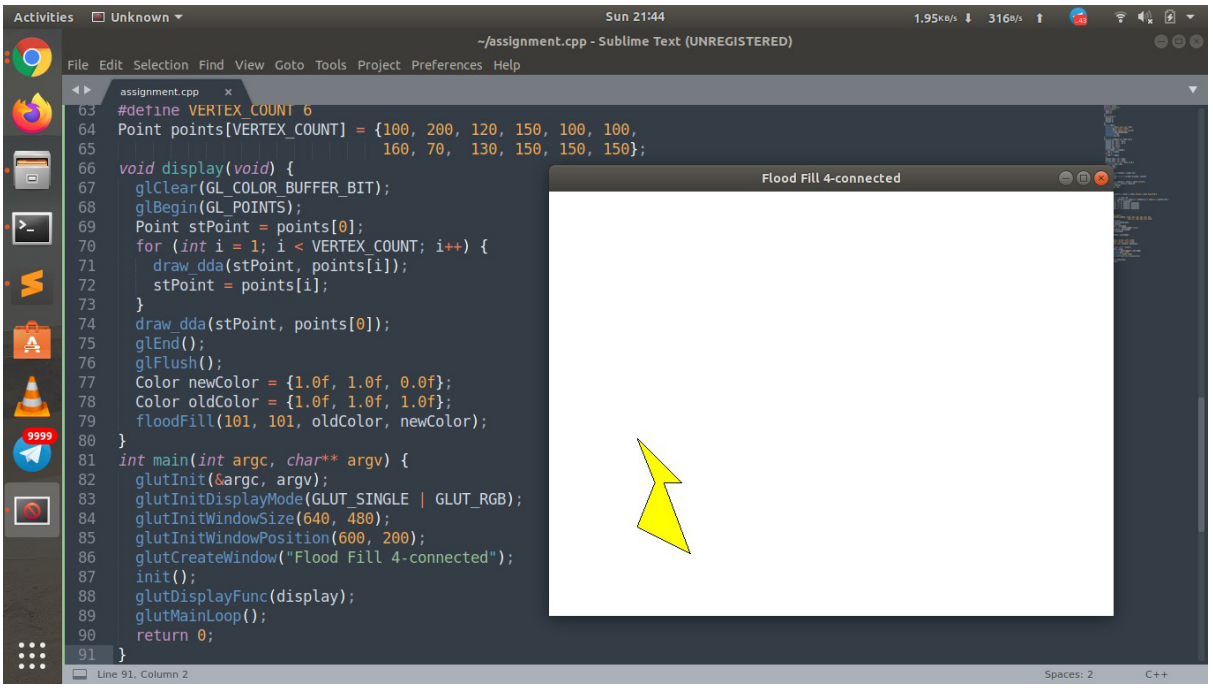

Activities
Unknown
Sun 21:41
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
#include <cmath>
#include <GL/glut.h>
#include <iostream>
using namespace std;
struct Point {
GLint x;
GLint y;
};
struct Color {
GLfloat r;
GLfloat g;
GLfloat b;
};
void init() {
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0f, 0.0f, 0.0f);
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
}
void draw_dda(Point p1, Point p2) {
GLfloat dx = p2.x - p1.x;
GLfloat dy = p2.y - p1.y;
GLfloat x1 = p1.x;
GLfloat y1 = p1.y;
GLfloat step = 0;
if (abs(dx) > abs(dy)) {
step = abs(dx);
} else {
Boundary Fill 8-connected
Line 2, Column 13
Spaces: 2
C++

Flood Fill 4 – connected concave

```
#include <cmath>
#include <GL/glut.h>
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
}; void
init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glPointSize(1.0f);
} void draw_dda(Point p1, Point
p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0; if
  (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else { step =
    abs(dy);
  }
  GLfloat xInc = dx / step;
  GLfloat yInc = dy / step; for
  (float i = 1; i <= step; i++) {
    glVertex2i(x1,
    y1); x1 += xInc;
    y1 += yInc;
  }
}
Color getPixelColor(GLint x, GLint y) {
  Color color;
  glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); return
  color;
} void setPixelColor(GLint x, GLint y, Color
color) {
```

```
glColor3f(color.r, color.g, color.b);
glBegin(GL_POINTS);
```

```
  glVertex2i(x, y);
  glEnd();
  glFlush();
} void floodFill(GLint x, GLint y, Color oldColor, Color
newColor) { Color color;
  color = getPixelColor(x, y); if (color.r == oldColor.r && color.g ==
  oldColor.g && color.b == oldColor.b) {
    setPixelColor(x, y, newColor);
    floodFill(x + 1, y, oldColor, newColor);
    floodFill(x, y + 1, oldColor, newColor);
    floodFill(x - 1, y, oldColor, newColor);
    floodFill(x, y - 1, oldColor, newColor);
  } return;
}
#define VERTEX_COUNT 6
Point points[VERTEX_COUNT] = {100, 200, 120, 150, 100, 100,
                        160, 70, 130, 150, 150, 150};
void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POINTS); Point stPoint =
  points[0]; for (int i = 1; i <
  VERTEX_COUNT; i++)
  { draw_dda(stPoint, points[i]); stPoint =
  points[i];
  }
  draw_dda(stPoint,
  points[0]); glEnd();
  glFlush();
  Color newColor = {1.0f, 1.0f, 0.0f};
  Color oldColor = {1.0f, 1.0f, 1.0f};
  floodFill(101, 101, oldColor,
  newColor);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(600, 200);
  glutCreateWindow("Flood Fill 4-connected");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

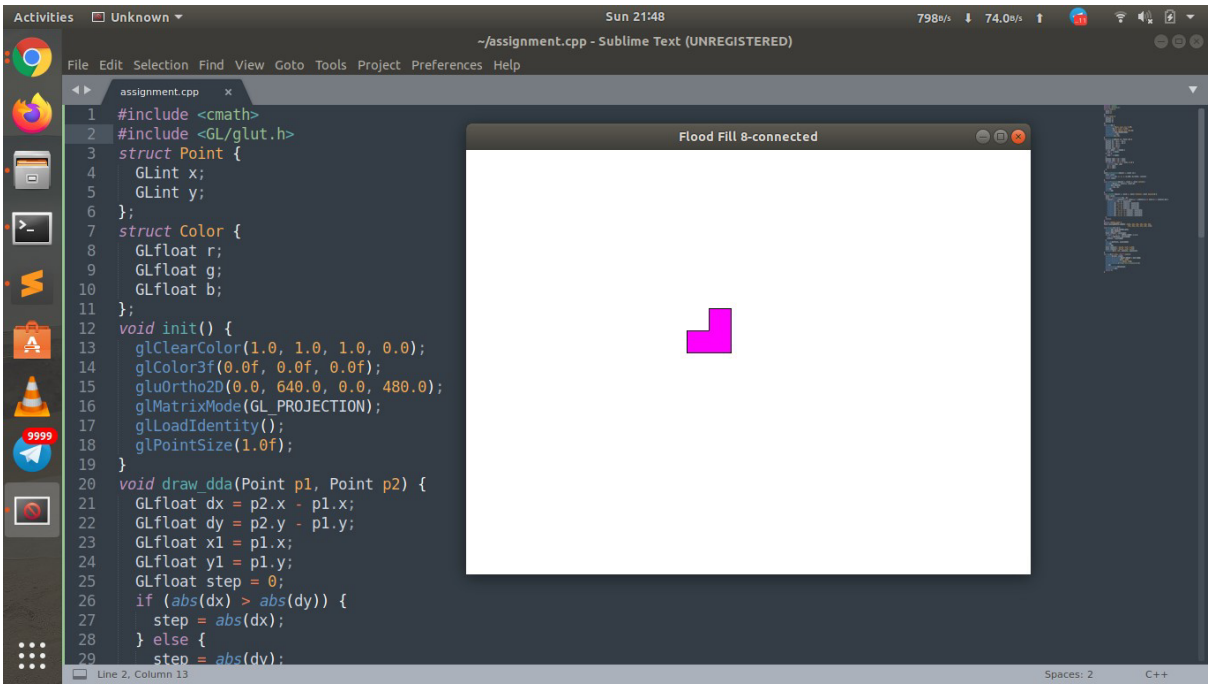

Activities
Unknown
Sun 21:44
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
#define VERTEX_COUNT 6
Point points[VERTEX_COUNT] = {100, 200, 120, 150, 100, 100,
160, 70, 130, 150, 150, 150};
void display(void) {
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
Point stPoint = points[0];
for (int i = 1; i < VERTEX_COUNT; i++) {
draw_dda(stPoint, points[i]);
stPoint = points[i];
}
draw_dda(stPoint, points[0]);
glEnd();
glFlush();
Color newColor = {1.0f, 1.0f, 0.0f};
Color oldColor = {1.0f, 1.0f, 1.0f};
floodFill(101, 101, oldColor, newColor);
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(640, 480);
glutInitWindowPosition(600, 200);
glutCreateWindow("Flood Fill 4-connected");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
Flood Fill 4-connected
Line 91, Column 2
Spaces: 2
C++

8 – connected concave

```
#include <cmath>
#include <GL/glut.h>
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
}; void
init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glPointSize(1.0f);
} void draw_dda(Point p1, Point
p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0; if
  (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else { step =
    abs(dy);
  }
  GLfloat xInc = dx / step;
  GLfloat yInc = dy / step; for
  (float i = 1; i <= step; i++) {
    glVertex2i(x1,
    y1); x1 += xInc;
    y1 += yInc;
  }
}
Color getPixelColor(GLint x, GLint y) {
  Color color;
  glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); return
  color;
} void setPixelColor(GLint x, GLint y, Color
color) {
  glColor3f(color.r, color.g, color.b);
  glBegin(GL_POINTS);
```

```
glVertex2i(x, y);
glEnd();
```

```
  glFlush();
} void floodFill(GLint x, GLint y, Color oldColor, Color
newColor) { Color color;
  color = getPixelColor(x, y); if (color.r == oldColor.r && color.g ==
  oldColor.g && color.b == oldColor.b) {
    setPixelColor(x, y, newColor); floodFill(x
    + 1, y, oldColor, newColor); floodFill(x, y
    + 1, oldColor, newColor); floodFill(x - 1,
    y, oldColor, newColor); floodFill(x, y - 1,
    oldColor, newColor); floodFill(x + 1, y +
    1, oldColor, newColor); floodFill(x - 1, y
    + 1, oldColor, newColor); floodFill(x - 1,
    y - 1, oldColor, newColor); floodFill(x +
    1, y - 1, oldColor, newColor);
  } return;
}
#define VERTEX_COUNT 6
Point points[VERTEX_COUNT] = {250, 250, 300, 250, 300, 300,
                        275, 300, 275, 275, 250, 275};
void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POINTS); Point stPoint =
  points[0]; for (int i = 1; i <
  VERTEX_COUNT; i++)
  { draw_dda(stPoint, points[i]); stPoint =
  points[i];
  }
  draw_dda(stPoint,
  points[0]); glEnd();
  glFlush();
  Color newColor = {1.0f, 0.0f, 1.0f};
  Color oldColor = {1.0f, 1.0f, 1.0f};
  floodFill(251, 251, oldColor,
  newColor);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(200, 200);
  glutCreateWindow("Flood Fill 8-connected");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

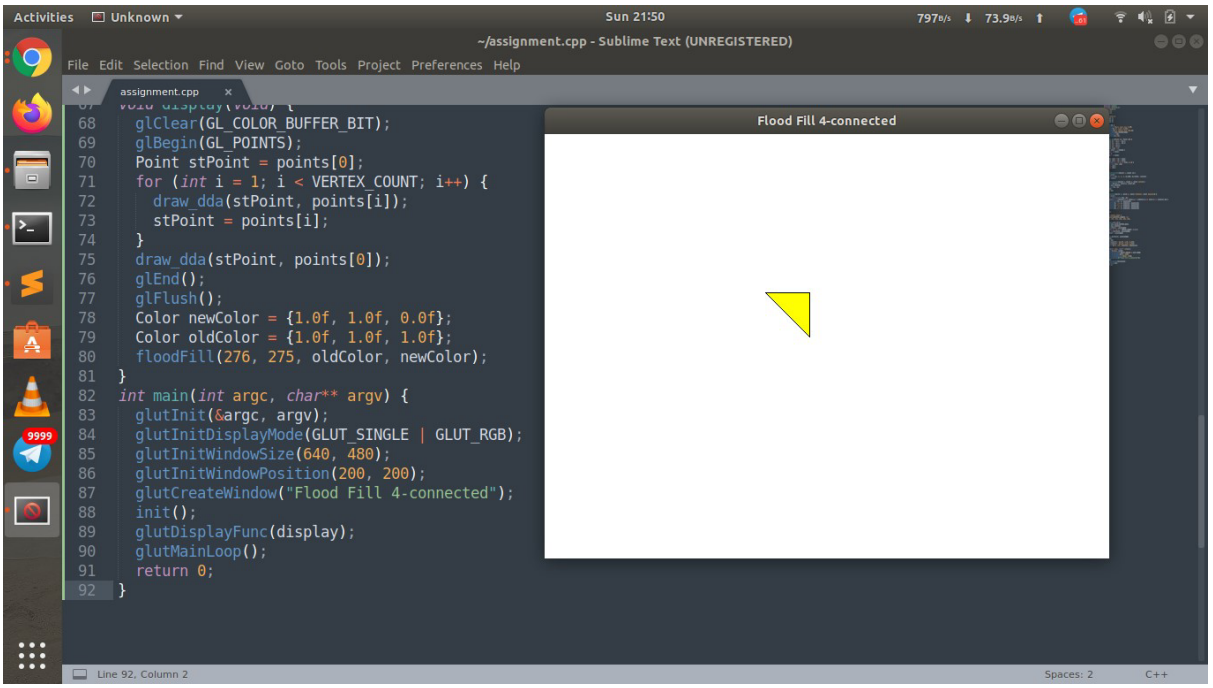

Activities
Unknown
Sun 21:48
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
#include <cmath>
#include <GL/glut.h>
struct Point {
GLint x;
GLint y;
};
struct Color {
GLfloat r;
GLfloat g;
GLfloat b;
};
void init() {
glClearColor(1.0, 1.0, 1.0, 0.0);
glColor3f(0.0f, 0.0f, 0.0f);
gluOrtho2D(0.0, 640.0, 0.0, 480.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glPointSize(1.0f);
}
void draw_dda(Point p1, Point p2) {
GLfloat dx = p2.x - p1.x;
GLfloat dy = p2.y - p1.y;
GLfloat x1 = p1.x;
GLfloat y1 = p1.y;
GLfloat step = 0;
if (abs(dx) > abs(dy)) {
step = abs(dx);
} else {
step = abs(dy);
Flood Fill 8-connected
Line 2, Column 13
Spaces: 2
C++

4 – connected convex

```
#include <cmath>
#include <GL/glut.h>
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
}; void
init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glPointSize(1.0f);
} void draw_dda(Point p1, Point
p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0; if
  (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else { step =
    abs(dy);
  }
  GLfloat xInc = dx / step;
  GLfloat yInc = dy / step; for
  (float i = 1; i <= step; i++) {
    glVertex2i(x1,
    y1); x1 += xInc;
    y1 += yInc;
  }
}
Color getPixelColor(GLint x, GLint y) {
  Color color;
  glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); return
  color;
} void setPixelColor(GLint x, GLint y, Color
color) {
  glColor3f(color.r, color.g, color.b);
  glBegin(GL_POINTS);
```

```
glVertex2i(x, y);
glEnd();
```

```
  glFlush();
} void floodFill(GLint x, GLint y, Color oldColor, Color
newColor) { Color color;
  color = getPixelColor(x, y); if (color.r == oldColor.r && color.g ==
  oldColor.g && color.b == oldColor.b) {
    setPixelColor(x, y, newColor);
    floodFill(x + 1, y, oldColor, newColor);
    floodFill(x, y + 1, oldColor, newColor);
    floodFill(x - 1, y, oldColor, newColor);
    floodFill(x, y - 1, oldColor, newColor);
  } return;
}
#define VERTEX_COUNT 3
Point points[VERTEX_COUNT] = {
    250, 300, 300, 300, 300, 250,
}; void
display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POINTS); Point stPoint =
  points[0]; for (int i = 1; i <
  VERTEX_COUNT; i++)
  { draw_dda(stPoint, points[i]); stPoint =
  points[i];
  }
  draw_dda(stPoint,
  points[0]); glEnd();
  glFlush();
  Color newColor = {1.0f, 1.0f, 0.0f};
  Color oldColor = {1.0f, 1.0f, 1.0f};
  floodFill(276, 275, oldColor,
  newColor);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(200, 200);
  glutCreateWindow("Flood Fill 4-connected");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

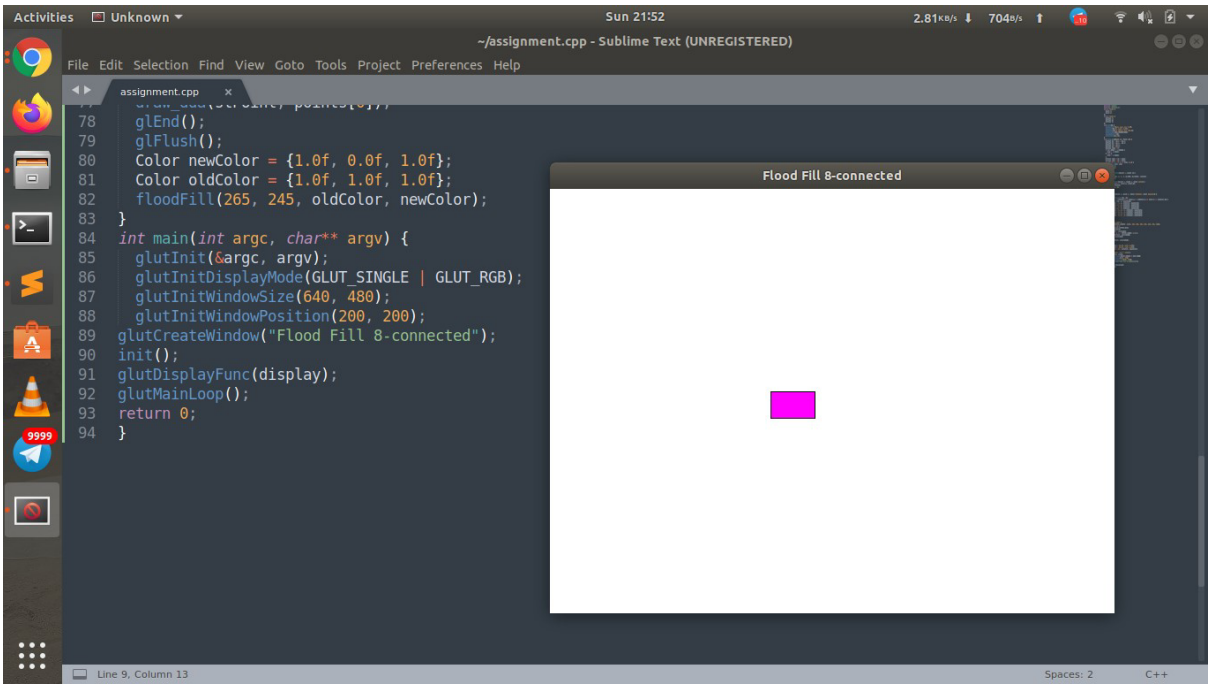

Activities
Unknown
Sun 21:50
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
Point stPoint = points[0];
for (int i = 1; i < VERTEX_COUNT; i++) {
draw_dda(stPoint, points[i]);
stPoint = points[i];
}
draw_dda(stPoint, points[0]);
glEnd();
glFlush();
Color newColor = {1.0f, 1.0f, 0.0f};
Color oldColor = {1.0f, 1.0f, 1.0f};
floodFill(276, 275, oldColor, newColor);
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(640, 480);
glutInitWindowPosition(200, 200);
glutCreateWindow("Flood Fill 4-connected");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
Flood Fill 4-connected
Line 92, Column 2
Spaces: 2
C++

8 – connected convex

```
#include <cmath>
#include <GL/glut.h>
struct Point {
  GLint x;
  GLint y;
};
struct Color {
  GLfloat r;
  GLfloat g;
  GLfloat b;
}; void
init() {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glColor3f(0.0f, 0.0f, 0.0f);
  gluOrtho2D(0.0, 640.0, 0.0, 480.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glPointSize(1.0f);
} void draw_dda(Point p1, Point
p2) {
  GLfloat dx = p2.x - p1.x;
  GLfloat dy = p2.y - p1.y;
  GLfloat x1 = p1.x;
  GLfloat y1 = p1.y;
  GLfloat step = 0; if
  (abs(dx) > abs(dy)) {
    step = abs(dx);
  } else { step =
    abs(dy);
  }
  GLfloat xInc = dx / step;
  GLfloat yInc = dy / step; for
  (float i = 1; i <= step; i++) {
    glVertex2i(x1,
    y1); x1 += xInc;
    y1 += yInc;
  }
}
Color getPixelColor(GLint x, GLint y) {
  Color color;
  glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color); return
  color;
} void setPixelColor(GLint x, GLint y, Color
color) {
  glColor3f(color.r, color.g, color.b);
  glBegin(GL_POINTS);
```

```
glVertex2i(x, y);
glEnd();
```

```
  glFlush();
} void floodFill(GLint x, GLint y, Color oldColor, Color
newColor) { Color color;
  color = getPixelColor(x, y); if (color.r == oldColor.r && color.g ==
  oldColor.g && color.b == oldColor.b) {
    setPixelColor(x, y, newColor); floodFill(x
    + 1, y, oldColor, newColor); floodFill(x, y
    + 1, oldColor, newColor); floodFill(x - 1,
    y, oldColor, newColor); floodFill(x, y - 1,
    oldColor, newColor); floodFill(x + 1, y +
    1, oldColor, newColor); floodFill(x + 1, y
    - 1, oldColor, newColor); floodFill(x - 1, y
    + 1, oldColor, newColor); floodFill(x - 1,
    y - 1, oldColor, newColor);
  } return;
}
#define VERTEX_COUNT 4
Point points[VERTEX_COUNT] = {250, 250, 300, 250, 300, 220, 250, 220};
void display(void) {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_POINTS); Point stPoint =
  points[0]; for (int i = 1; i <
  VERTEX_COUNT; i++)
  { draw_dda(stPoint, points[i]); stPoint =
  points[i];
  }
  draw_dda(stPoint,
  points[0]); glEnd();
  glFlush();
  Color newColor = {1.0f, 0.0f, 1.0f};
  Color oldColor = {1.0f, 1.0f, 1.0f};
  floodFill(265, 245, oldColor,
  newColor);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(200, 200);
glutCreateWindow("Flood Fill 8-connected");
init();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```


Activities
Unknown
Sun 21:52
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
78 glEnd();
79 glFlush();
80 Color newColor = {1.0f, 0.0f, 1.0f};
81 Color oldColor = {1.0f, 1.0f, 1.0f};
82 floodFill(265, 245, oldColor, newColor);
83 }
84 int main(int argc, char** argv) {
85 glutInit(&argc, argv);
86 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
87 glutInitWindowSize(640, 480);
88 glutInitWindowPosition(200, 200);
89 glutCreateWindow("Flood Fill 8-connected");
90 init();
91 glutDisplayFunc(display);
92 glutMainLoop();
93 return 0;
94 }
Flood Fill 8-connected
Line 9, Column 13
Spaces: 2
C++

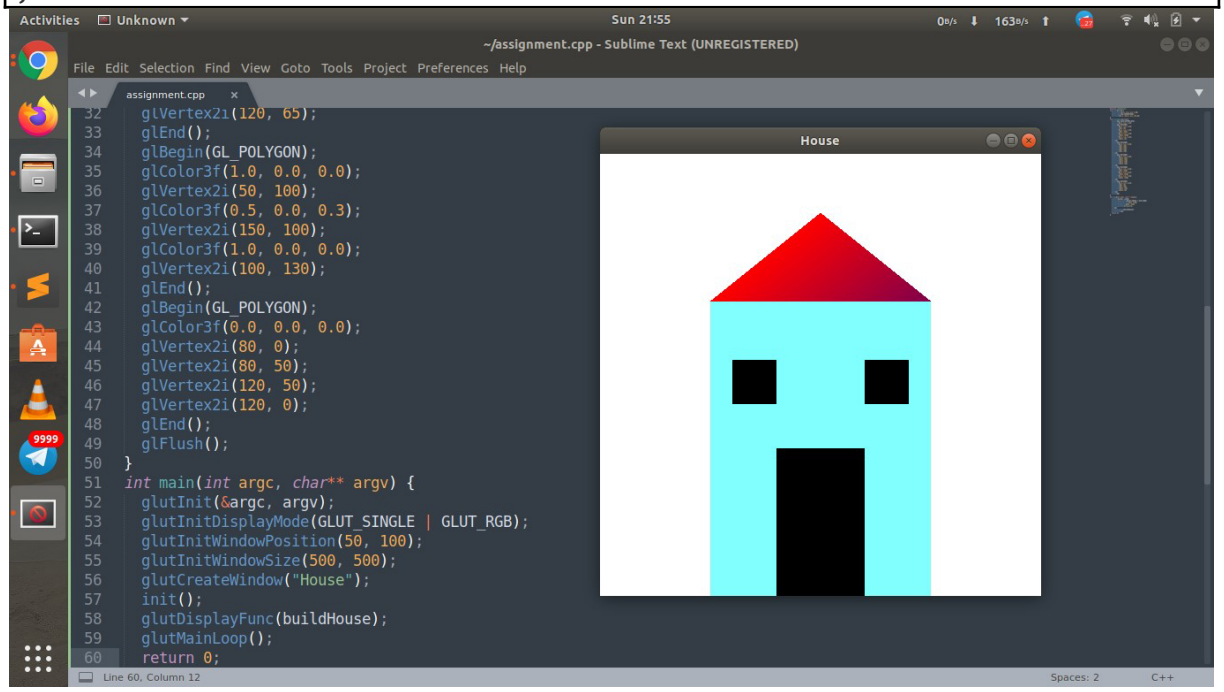
Q6: Write a program for drawing the following simple two dimensional objects using certain graphic functions available for drawing lines, rectangles, polygons, ellipses & circles which generates pixel activation list.

(i) House

```
#include <GL/glut.h>
void init(void) {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0.0, 200.0, 0.0, 150.0);
} void buildHouse(void)
{ glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glBegin(GL_POLYGON);
  glColor3f(0.5, 1.0, 1.0);
  glVertex2i(50, 0);
  glColor3f(0.5, 1.0, 1.0);
  glVertex2i(50, 100);
  glColor3f(0.5, 1.0, 1.0);
  glVertex2i(150, 100);
  glColor3f(0.5, 1.0, 1.0);
  glVertex2i(150, 0);
  glEnd();
  glBegin(GL_POLYGON);
  glColor3f(0.0, 0.0, 0.0);
  glVertex2i(60, 80);
  glVertex2i(80, 80);
  glVertex2i(80, 65);
  glVertex2i(60, 65);
  glEnd();
  glBegin(GL_POLYGON);
  glColor3f(0.0, 0.0, 0.0);
  glVertex2i(120, 80);
  glVertex2i(140, 80);
  glVertex2i(140, 65);
  glVertex2i(120, 65);
  glEnd();
  glBegin(GL_POLYGON);
  glColor3f(1.0, 0.0, 0.0);
  glVertex2i(50, 100);
  glColor3f(0.5, 0.0, 0.3);
  glVertex2i(150, 100);
  glColor3f(1.0, 0.0, 0.0);
  glVertex2i(100, 130);
  glEnd();
  glBegin(GL_POLYGON);
```

```
glColor3f(0.0, 0.0, 0.0);
glVertex2i(80, 0);
```

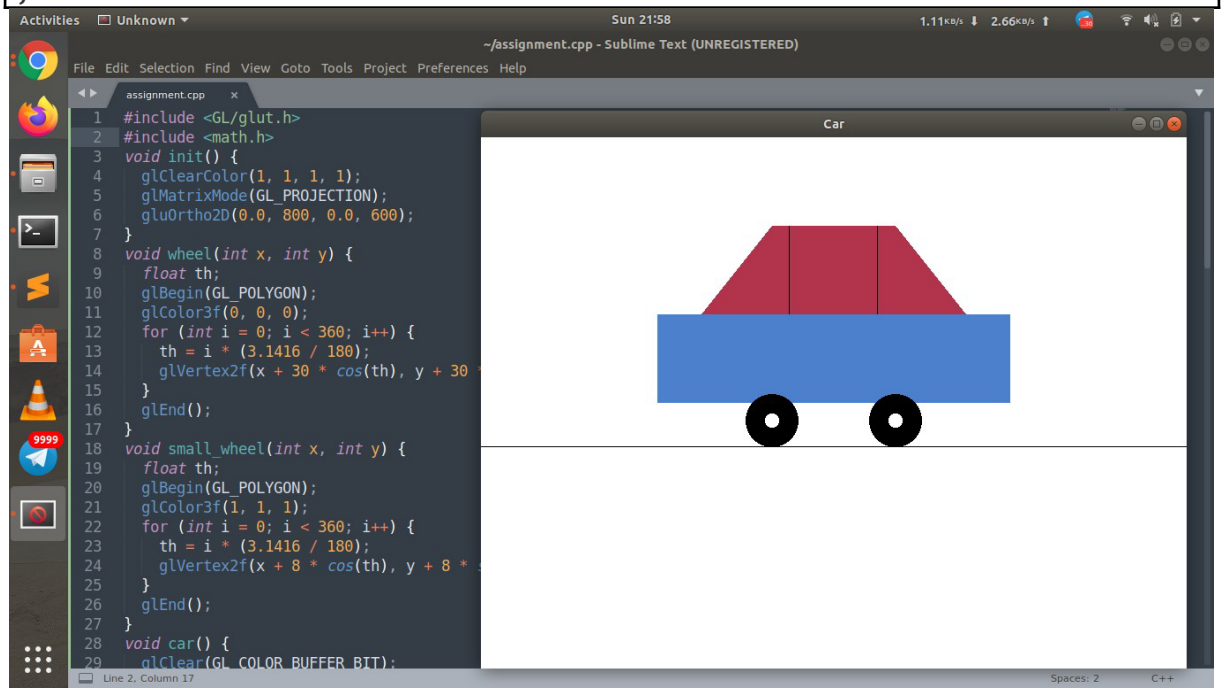
```
  glVertex2i(80, 50);
  glVertex2i(120, 50);
  glVertex2i(120, 0);
  glEnd();
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(50, 100);
  glutInitWindowSize(500, 500);
  glutCreateWindow("House");
  init();
  glutDisplayFunc(buildHouse);
  glutMainLoop();
  return 0;
}
```



(ii) Car

```
#include <GL/glut.h>
#include <math.h>
void init() {
  glClearColor(1, 1, 1, 1);
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0.0, 800, 0.0, 600);
} void wheel(int x, int
y) {
  float th;
  glBegin(GL_POLYGON);
  glColor3f(0, 0, 0); for (int i
  = 0; i < 360; i++) {
    th = i * (3.1416 / 180); glVertex2f(x + 30
    * cos(th), y + 30 * sin(th));
  } glEnd(); } void
small_wheel(int x, int y) {
  float th;
  glBegin(GL_POLYGON);
  glColor3f(1, 1, 1); for (int i
  = 0; i < 360; i++) {
    th = i * (3.1416 / 180); glVertex2f(x + 8
    * cos(th), y + 8 * sin(th));
  } glEnd();
 } void
car() {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.3, 0.5, 0.8);
  glBegin(GL_POLYGON);
  glVertex2f(200, 300);
  glVertex2f(200, 400);
  glVertex2f(600, 400);
  glVertex2f(600, 300);
  glEnd();
  glColor3f(0.7, 0.2, 0.3);
  glBegin(GL_POLYGON);
  glVertex2f(250, 400);
  glVertex2f(330, 500);
  glVertex2f(470, 500);
  glVertex2f(550, 400);
  glEnd(); wheel(330,
  280); wheel(470, 280);
  small_wheel(330, 280);
  small_wheel(470, 280);
  glColor3f(0, 0, 0);
```

```
  glBegin(GL_LINES);
  glVertex2f(0, 250);
  glVertex2f(800,
  250); glEnd();
  glColor3f(0, 0, 0);
  glBegin(GL_LINES);
  glVertex2f(350,
  500);
  glVertex2f(350,
  400); glEnd();
  glColor3f(0, 0, 0);
  glBegin(GL_LINES);
  glVertex2f(450,
  500);
  glVertex2f(450,
  400); glEnd();
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Car");
  init();
  glutDisplayFunc(car);
  glutMainLoop();
}
```



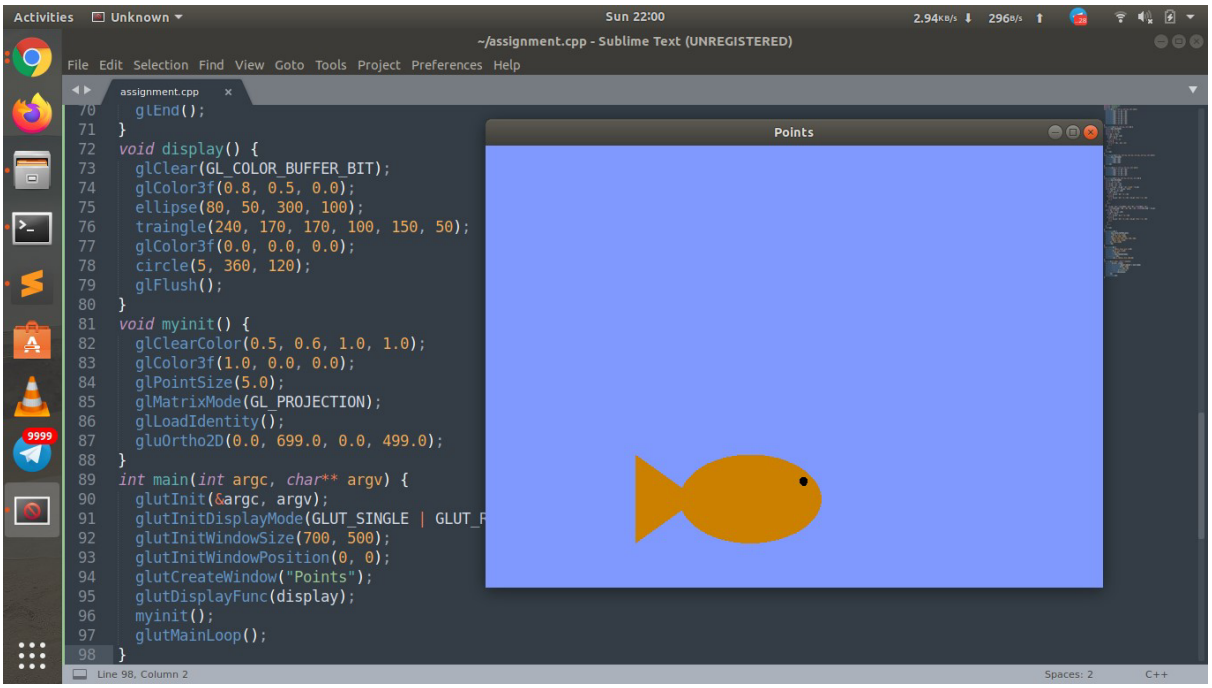
(iii) Fish

```
#include <GL/glut.h> #include
<math.h> void draw(int x, int y, int
xc, int yc) {
  glVertex2f(xc + x, yc + y);
  glVertex2f(xc + x, yc - y);
  glVertex2f(xc - x, yc - y);
  glVertex2f(xc - x, yc + y);
  glVertex2f(xc + y, yc + x);
  glVertex2f(xc - y, yc + x);
  glVertex2f(xc - y, yc - x);
  glVertex2f(xc + y, yc - x);
}
void circle(int R, int xc, int yc) {
  glBegin(GL_POLYGON);
  int x = 0, y = R; int P = 3
  - 2 * R; while (y >= x) {
    draw(x, y, xc, yc);
    if (P < 0) {
      P += 4 * x + 6;
    } else {
      P += 4 * (x - y) + 10;
      y--;
    } x++; } glEnd(); } void traingle(int x1, int x2, int x3,
int y1, int y2, int y3) {
  glBegin(GL_POLYGON);
  glVertex2f(x1, y1); glVertex2f(x2,
y2); glVertex2f(x3, y3); glEnd(); }
void draw1(int x, int y, int xc, int yc)
{
  glVertex2f(xc + x, yc + y);
  glVertex2f(xc + x, yc - y);
  glVertex2f(xc - x, yc - y);
  glVertex2f(xc - x, yc + y);
} void ellipse(int rx, int ry, int xc, int
yc) {
  glBegin(GL_POLYGON); int x = 0, y =
  ry; int ry_sq = ry * ry; int rx_sq = rx *
  rx; int P = ry_sq - rx_sq * ry + 0.25 *
  rx_sq; while (ry_sq * x < rx_sq * y) {
```

```
    draw1(x, y, xc, yc);
    if (P < 0) {
      P += ry_sq * (2 * x + 3);
    } else {
      P += ry_sq * (2 * x + 3) + rx_sq * (-2 * y + 2); y-
      -;
    }
    x++;
  }
  P = ry_sq * (x + float(1) / 2) * (x + float(1) / 2) +
      double(rx_sq) * (y - 1) * (y - 1) - double(rx_sq) * ry_sq;
  while (y >= 0) {
    draw1(x, y, xc, yc);
    if (P >= 0) {
      P += rx_sq * (-2 * y + 3);
    } else {
      P += ry_sq * (2 * x + 2) + rx_sq * (-2 * y + 3);
      x++;
    }
    y--;
  } glEnd(); }
void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.8, 0.5, 0.0); ellipse(80,
  50, 300, 100); traingle(240, 170,
  170, 100, 150, 50);
  glColor3f(0.0, 0.0, 0.0);
  circle(5, 360, 120);
  glFlush();
} void
myinit() {
  glClearColor(0.5, 0.6, 1.0, 1.0);
  glColor3f(1.0, 0.0, 0.0);
  glPointSize(5.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0, 699.0, 0.0,
  499.0);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(700, 500);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("Points");
  glutDisplayFunc(display);
```

```
myinit();
glutMainLoop();
```

```
}
```


Activities
Unknown
Sun 22:00
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
glEnd();
}
void display() {
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.8, 0.5, 0.0);
ellipse(80, 50, 300, 100);
traingle(240, 170, 170, 100, 150, 50);
glColor3f(0.0, 0.0, 0.0);
circle(5, 360, 120);
glFlush();
}
void myinit() {
glClearColor(0.5, 0.6, 1.0, 1.0);
glColor3f(1.0, 0.0, 0.0);
glPointSize(5.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 699.0, 0.0, 499.0);
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_F
glutInitWindowSize(700, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("Points");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
Points
Line 98, Column 2
Spaces: 2
C++

(iv) Man

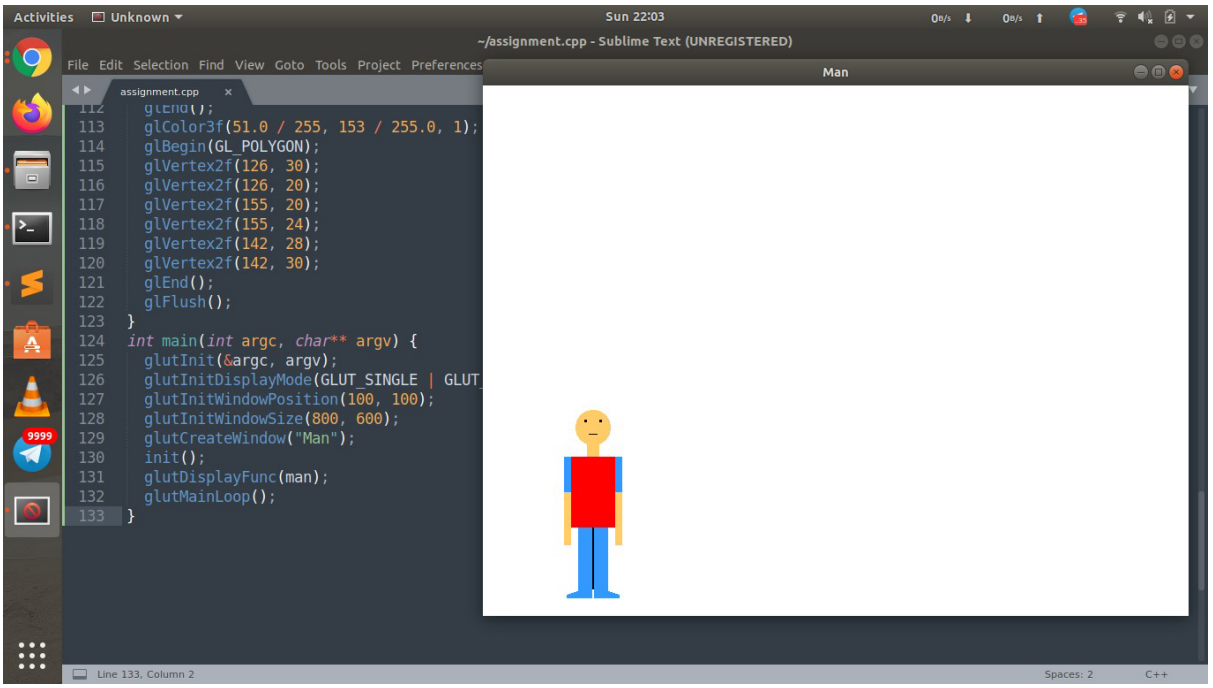
```
#include <GL/glut.h>
#include <math.h>
void init() {
  glClearColor(1, 1, 1, 1);
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0.0, 800, 0.0, 600);
} void circle(int x, int
y) {
  float th; glColor3f(1, 204 / 255.0,
  104 / 255.0);
  glBegin(GL_POLYGON);
  for (int i = 0; i < 360; i++) {
    th = i * (3.1416 / 180); glVertex2f(x + 20
    * cos(th), y + 20 * sin(th));
  } glEnd(); }
void man()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1, 0, 0);
  glBegin(GL_POLYGON);
  glVertex2f(100, 100);
  glVertex2f(150, 100);
  glVertex2f(150, 180);
  glVertex2f(100, 180); glEnd();
  glColor3f(51.0 / 255, 153 / 255.0,
  1);
  glBegin(GL_POLYGON);
  glVertex2f(108, 100);
  glVertex2f(108, 30);
  glVertex2f(124, 30);
  glVertex2f(124, 100); glEnd();
  glColor3f(0, 0, 0);
  glBegin(GL_POLYGON);
  glVertex2f(124, 100);
  glVertex2f(124, 30);
  glVertex2f(126, 30);
  glVertex2f(126, 100); glEnd();
  glColor3f(51.0 / 255, 153 / 255.0,
  1);
  glBegin(GL_POLYGON);
  glVertex2f(100, 180);
  glVertex2f(92, 180);
  glVertex2f(92, 140);
  glVertex2f(100, 140);
  glEnd();
```

```
glColor3f(1, 204 / 255.0, 104 / 255.0);
```

```
glBegin(GL_POLYGON);
glVertex2f(92, 140);
glVertex2f(92, 80);
glVertex2f(100, 80);
glVertex2f(100, 140); glEnd();
glColor3f(51.0 / 255, 153 / 255.0,
1);
glBegin(GL_POLYGON);
glVertex2f(150, 180);
glVertex2f(158, 180);
glVertex2f(158, 140);
glVertex2f(150, 140); glEnd();
glColor3f(1, 204 / 255.0, 104 /
255.0);
glBegin(GL_POLYGON);
glVertex2f(158, 140);
glVertex2f(158, 80);
glVertex2f(150, 80);
glVertex2f(150, 140); glEnd();
glColor3f(51.0 / 255, 153 / 255.0,
1);
glBegin(GL_POLYGON);
glVertex2f(126, 100);
glVertex2f(126, 30);
glVertex2f(142, 30);
glVertex2f(142, 100); glEnd();
glColor3f(1, 204 / 255.0, 104 /
255.0);
glBegin(GL_POLYGON);
glVertex2f(118, 180);
glVertex2f(118, 195);
glVertex2f(132, 195);
glVertex2f(132, 180);
glEnd(); circle(125,
213); glColor3f(0, 0, 0);
glBegin(GL_LINES);
glLineWidth(5);
glVertex2f(120, 205);
glVertex2f(130, 205);
glEnd(); glColor3f(0, 0,
0);
glBegin(GL_POLYGON);
glVertex2f(132, 222);
glVertex2f(135, 222);
glVertex2f(135, 219);
glVertex2f(132, 219);
glEnd(); glColor3f(0, 0,
```

```
0);
```

```
  glBegin(GL_POLYGON);
  glVertex2f(118, 222);
  glVertex2f(115, 222);
  glVertex2f(115, 219);
  glVertex2f(118, 219); glEnd();
  glColor3f(51.0 / 255, 153 / 255.0,
  1);
  glBegin(GL_POLYGON);
  glVertex2f(124, 30);
  glVertex2f(124, 20);
  glVertex2f(95, 20); glVertex2f(95,
  24); glVertex2f(108, 28);
  glVertex2f(108, 30); glEnd();
  glColor3f(51.0 / 255, 153 / 255.0,
  1);
  glBegin(GL_POLYGON);
  glVertex2f(126, 30);
  glVertex2f(126, 20);
  glVertex2f(155, 20);
  glVertex2f(155, 24);
  glVertex2f(142, 28);
  glVertex2f(142, 30);
  glEnd();
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Man");
  init();
  glutDisplayFunc(man);
  glutMainLoop();
}
```

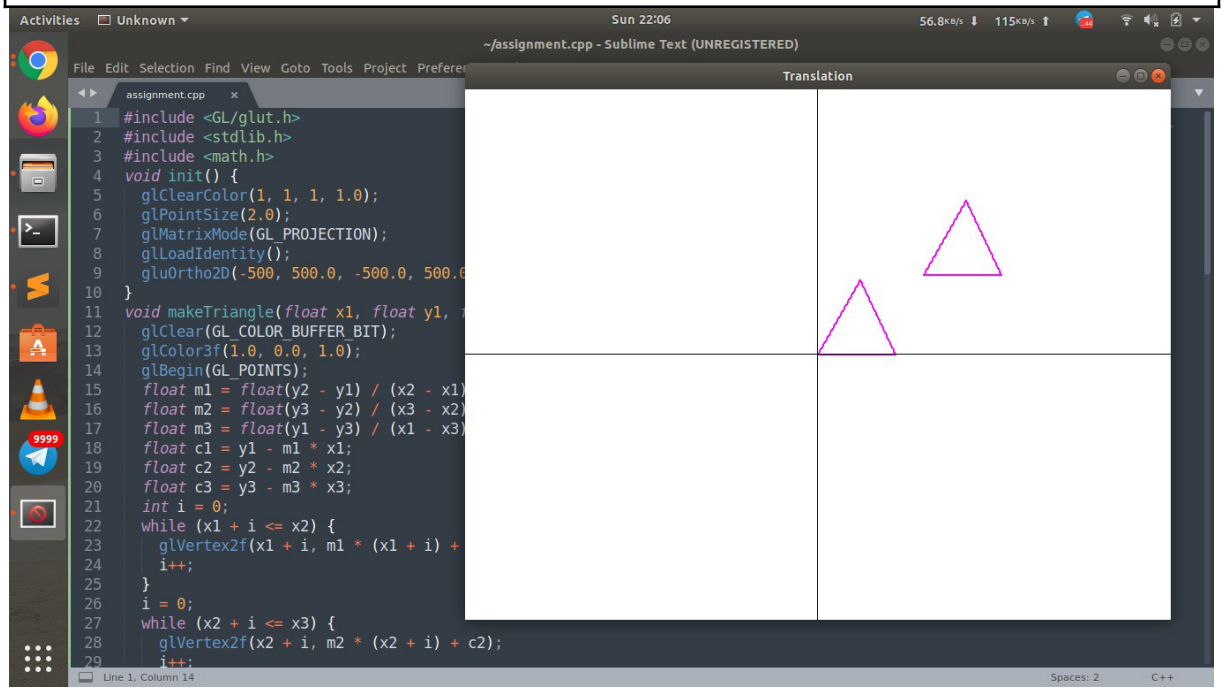

Activities
Unknown
Sun 22:03
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences
assignment.cpp
glColor3f(51.0 / 255, 153 / 255.0, 1);
glBegin(GL_POLYGON);
glVertex2f(126, 30);
glVertex2f(126, 20);
glVertex2f(155, 20);
glVertex2f(155, 24);
glVertex2f(142, 28);
glVertex2f(142, 30);
glEnd();
glFlush();
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT
glutInitWindowPosition(100, 100);
glutInitWindowSize(800, 600);
glutCreateWindow("Man");
init();
glutDisplayFunc(man);
glutMainLoop();
}
Man
Line 133, Column 2
Spaces: 2
C++

Q7: Write a program to perform basic 2D transformation (translation, rotation and scaling) about origin and about a fixed point without using direct OpenGl functions for the transformations.

Translation:

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
void init() {
  glClearColor(1, 1, 1, 1.0);
  glPointSize(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); gluOrtho2D(-500,
  500.0, -500.0, 500.0);
} void makeTriangle(float x1, float y1, float x2, float y2, float x3,
float y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 0.0, 1.0);
  glBegin(GL_POINTS); float m1 =
  float(y2 - y1) / (x2 - x1); float m2
  = float(y3 - y2) / (x3 - x2); float
  m3 = float(y1 - y3) / (x1 - x3);
  float c1 = y1 - m1 *
  x1; float c2 = y2 - m2
  * x2; float c3 = y3 -
  m3 * x3; int i = 0;
  while (x1 + i <= x2) {
    glVertex2f(x1 + i, m1 * (x1 + i) +
  c1); i++; } i = 0; while (x2 + i <= x3) {
    glVertex2f(x2 + i, m2 * (x2 + i) +
  c2); i++; } i = 0;
  while (x3 - i >= x1) {
    glVertex2f(x3 - i, m3 * (x3 - i) + c3);
    i++;
  }
} void translate(float x1, float y1, float x2, float y2, float x3,
float y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  x1 = x1 + 150;
  x2 = x2 + 150;
  x3 = x3 + 150;
  y1 = y1 + 150;
  y2 = y2 + 150;
  y3 = y3 + 150;
```

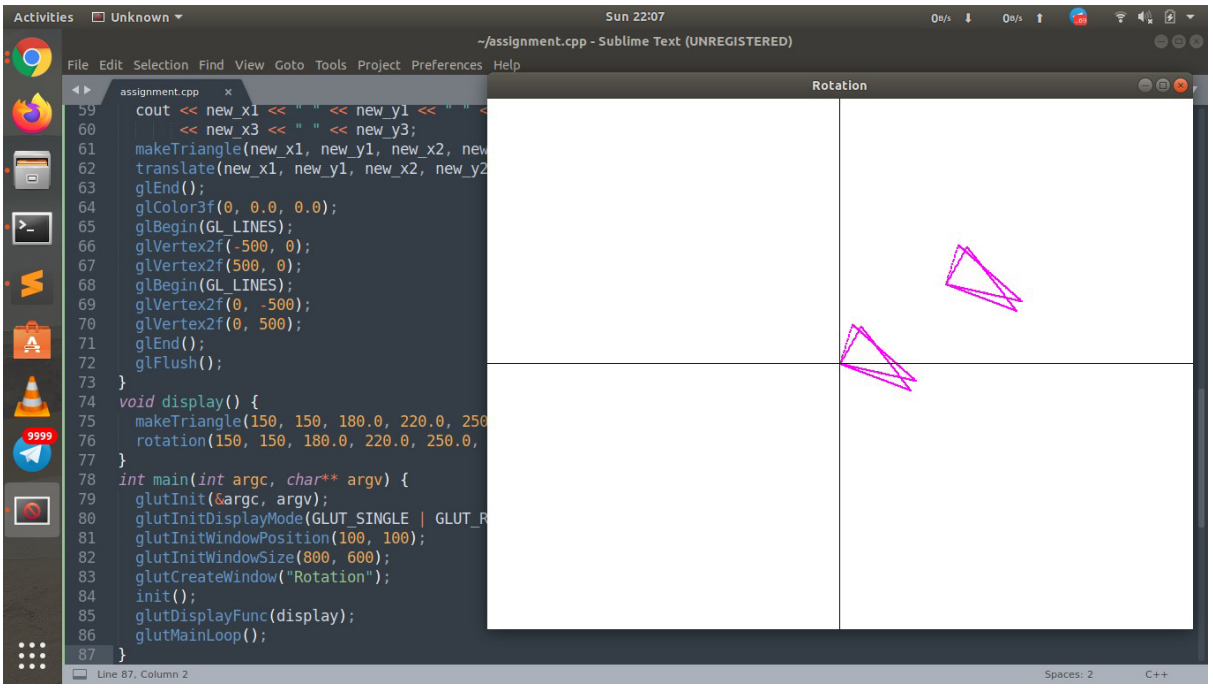
```
  makeTriangle(x1, y1, x2, y2, x3,
  y3); glEnd(); glColor3f(0, 0.0, 0.0);
  glBegin(GL_LINES); glVertex2f(-
  500, 0); glVertex2f(500, 0);
  glBegin(GL_LINES); glVertex2f(0,
  -500); glVertex2f(0, 500); glEnd();
  glFlush();
} void
display() {
  makeTriangle(0, 0, 60.0, 140.0, 110.0, 0.0);
  translate(0, 0, 60.0, 140.0, 110.0, 0.0);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Translation");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```



Rotation:

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
using namespace std;
void init() {
  glClearColor(1, 1, 1, 1.0);
  glPointSize(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); gluOrtho2D(-500,
  500.0, -500.0, 500.0);
} void makeTriangle(float x1, float y1, float x2, float y2, float x3,
float y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 0.0, 1.0);
  glBegin(GL_POINTS); float m1 =
  float(y2 - y1) / (x2 - x1); float m2
  = float(y3 - y2) / (x3 - x2); float
  m3 = float(y1 - y3) / (x1 - x3);
  float c1 = y1 - m1 *
  x1; float c2 = y2 - m2
  * x2; float c3 = y3 -
  m3 * x3; int i = 0;
  while (x1 + i <= x2) {
    glVertex2f(x1 + i, m1 * (x1 + i) +
  c1); i++; } i = 0; while (x2 + i <= x3) {
    glVertex2f(x2 + i, m2 * (x2 + i) +
  c2); i++; } i = 0;
  while (x3 - i >= x1) {
    glVertex2f(x3 - i, m3 * (x3 - i) + c3);
    i++;
  }
}
void translate(float x1, float y1, float x2, float y2, float x3, float y3,
            float xp, float yp) {
  glClear(GL_COLOR_BUFFER_BIT);
  x1 = x1 + xp;
  x2 = x2 + xp;
  x3 = x3 + xp;
  y1 = y1 + yp;
  y2 = y2 + yp;
  y3 = y3 + yp;
  makeTriangle(x1, y1, x2, y2, x3, y3);
```

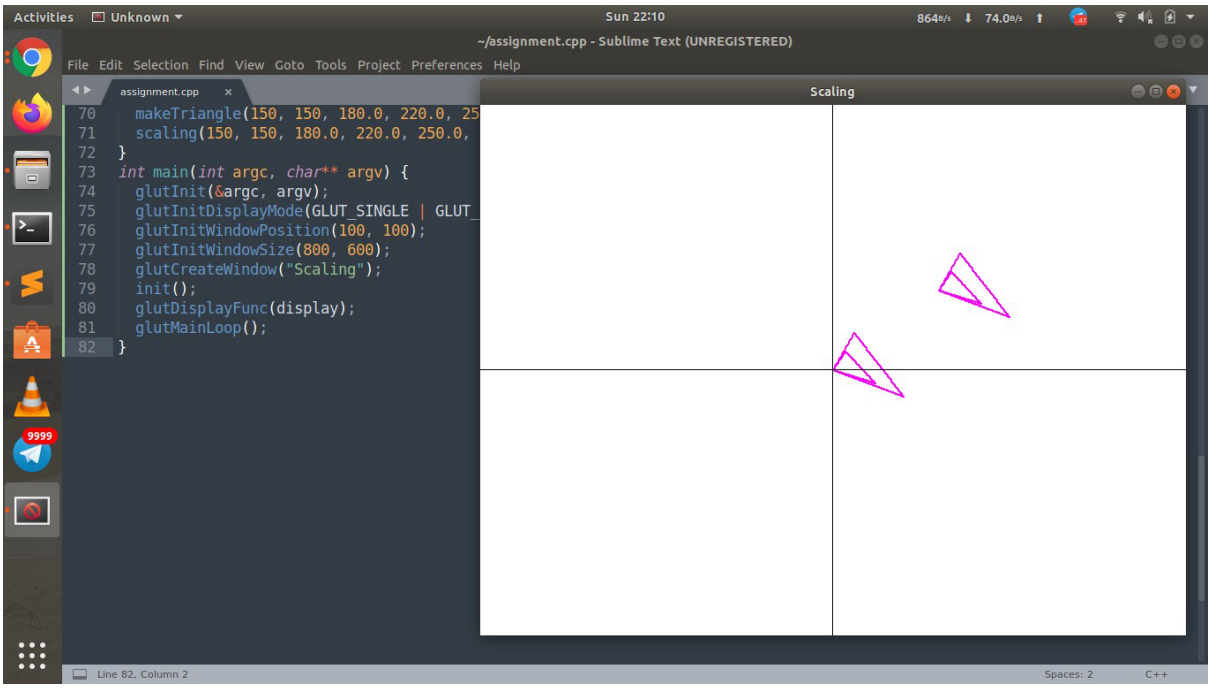
```
} void rotation(float x1, float y1, float x2, float y2, float x3, float
y3) {
  translate(x1, y1, x2, y2, x3, y3, -x1, -y1); float th =
  10 * 3.14159 / 180; float new_x1 = (x1 - x1) *
  cos(th) - (y1 - y1) * sin(th); float new_y1 = (x1 - x1)
  * sin(th) + (y1 - y1) * cos(th); float new_x2 = (x2 -
  x1) * cos(th) - (y2 - y1) * sin(th); float new_y2 =
  (x2 - x1) * sin(th) + (y2 - y1) * cos(th); float
  new_x3 = (x3 - x1) * cos(th) - (y3 - y1) * sin(th);
  float new_y3 = (x3 - x1) * sin(th) + (y3 - y1) *
  cos(th);
  cout << new_x1 << " " << new_y1 << " " << new_x2 << " " << new_y2 << " "
      << new_x3 << " " << new_y3; makeTriangle(new_x1, new_y1,
  new_x2, new_y2, new_x3, new_y3); translate(new_x1, new_y1,
  new_x2, new_y2, new_x3, new_y3, 150, 150); glEnd(); glColor3f(0,
  0.0, 0.0); glBegin(GL_LINES); glVertex2f(-500, 0); glVertex2f(500, 0);
  glBegin(GL_LINES); glVertex2f(0, -500); glVertex2f(0, 500); glEnd();
  glFlush();
} void
display() {
  makeTriangle(150, 150, 180.0, 220.0, 250.0, 100.0);
  rotation(150, 150, 180.0, 220.0, 250.0, 100.0);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Rotation");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```


Activities
Unknown
Sun 22:07
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
cout << new_x1 << " " << new_y1 << " "
<< new_x3 << " " << new_y3;
makeTriangle(new_x1, new_y1, new_x2, new
translate(new_x1, new_y1, new_x2, new_y2
glEnd();
glColor3f(0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2f(-500, 0);
glVertex2f(500, 0);
glBegin(GL_LINES);
glVertex2f(0, -500);
glVertex2f(0, 500);
glEnd();
glFlush();
}
void display() {
makeTriangle(150, 150, 180.0, 220.0, 250
rotation(150, 150, 180.0, 220.0, 250.0,
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_R
glutInitWindowPosition(100, 100);
glutInitWindowSize(800, 600);
glutCreateWindow("Rotation");
init();
glutDisplayFunc(display);
glutMainLoop();
}
Rotation
Line 87, Column 2
Spaces: 2
C++

Scaling:

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
void init() {
  glClearColor(1, 1, 1, 1.0);
  glPointSize(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); gluOrtho2D(-500,
  500.0, -500.0, 500.0);
} void makeTriangle(float x1, float y1, float x2, float y2, float x3,
float y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 0.0, 1.0);
  glBegin(GL_POINTS); float m1 =
  float(y2 - y1) / (x2 - x1); float m2
  = float(y3 - y2) / (x3 - x2); float
  m3 = float(y1 - y3) / (x1 - x3);
  float c1 = y1 - m1 *
  x1; float c2 = y2 - m2
  * x2; float c3 = y3 -
  m3 * x3; int i = 0;
  while (x1 + i <= x2) {
    glVertex2f(x1 + i, m1 * (x1 + i) +
  c1); i++; } i = 0; while (x2 + i <= x3) {
    glVertex2f(x2 + i, m2 * (x2 + i) +
  c2); i++; } i = 0; while (x3 - i >= x1) {
    glVertex2f(x3 - i, m3 * (x3 - i) + c3);
    i++;
  }
}
void translate(float x1, float y1, float x2, float y2, float x3, float y3,
               float xp, float yp) {
  glClear(GL_COLOR_BUFFER_BIT);
  x1 = x1 + xp;
  x2 = x2 + xp;
  x3 = x3 + xp;
  y1 = y1 + yp;
  y2 = y2 + yp;
  y3 = y3 + yp;
  makeTriangle(x1, y1, x2, y2, x3, y3);
}
void scaling(float x1, float y1, float x2, float y2, float x3, float y3) {
```

```
  translate(x1, y1, x2, y2, x3, y3, -x1, -y1); float new_x1 = (x1 - x1) *
  0.60; float new_y1 = (y1 - y1) * 0.5; float new_x2 = (x2 - x1) * 0.60;
  float new_y2 = (y2 - y1) * 0.5; float new_x3 = (x3 - x1) * 0.60; float
  new_y3 = (y3 - y1) * 0.5; makeTriangle(new_x1, new_y1, new_x2,
  new_y2, new_x3, new_y3); translate(new_x1, new_y1, new_x2,
  new_y2, new_x3, new_y3, 150, 150); glEnd(); glColor3f(0, 0.0, 0.0);
  glBegin(GL_LINES); glVertex2f(-500, 0); glVertex2f(500, 0);
  glBegin(GL_LINES); glVertex2f(0, -500); glVertex2f(0, 500); glEnd();
  glFlush();
} void
display() {
  makeTriangle(150, 150, 180.0, 220.0, 250.0, 100.0);
  scaling(150, 150, 180.0, 220.0, 250.0, 100.0);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Scaling");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```


Activities
Unknown
Sun 22:10
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
70 makeTriangle(150, 150, 180.0, 220.0, 25
71 scaling(150, 150, 180.0, 220.0, 250.0,
72 }
73 int main(int argc, char** argv) {
74 glutInit(&argc, argv);
75 glutInitDisplayMode(GLUT_SINGLE | GLUT_
76 glutInitWindowPosition(100, 100);
77 glutInitWindowSize(800, 600);
78 glutCreateWindow("Scaling");
79 init();
80 glutDisplayFunc(display);
81 glutMainLoop();
82 }
Scaling
Line 82, Column 2
Spaces: 2
C++

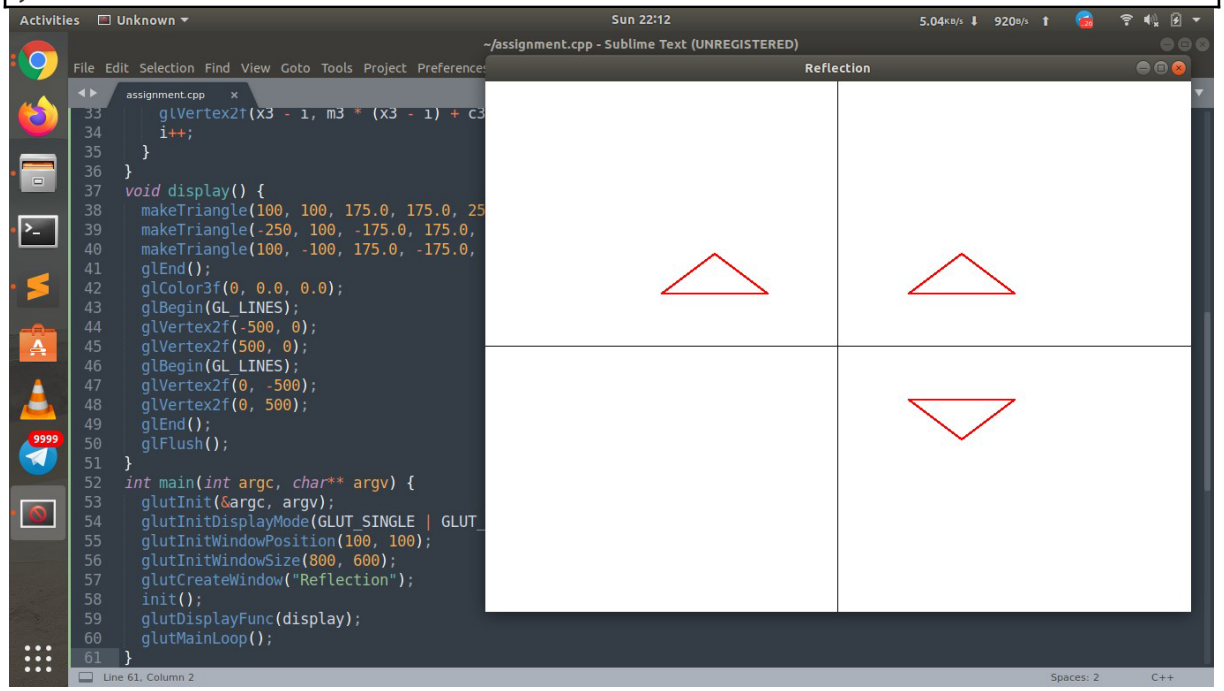
Q8: Write a program to perform:

(i) Reflection about x-axis, y-axis and a line y = x+2

About x and y axis:

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
void init() {
  glClearColor(1, 1, 1, 1.0);
  glPointSize(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); gluOrtho2D(-500,
  500.0, -500.0, 500.0);
} void makeTriangle(float x1, float y1, float x2, float y2, float x3,
float y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 0.0, 0.0);
  glBegin(GL_POINTS); float m1 =
  float(y2 - y1) / (x2 - x1); float m2
  = float(y3 - y2) / (x3 - x2); float
  m3 = float(y1 - y3) / (x1 - x3);
  float c1 = y1 - m1 *
  x1; float c2 = y2 - m2
  * x2; float c3 = y3 -
  m3 * x3; int i = 0;
  while (x1 + i <= x2) {
    glVertex2f(x1 + i, m1 * (x1 + i) +
  c1); i++; } i = 0; while (x2 + i <= x3) {
    glVertex2f(x2 + i, m2 * (x2 + i) +
  c2); i++; } i = 0;
  while (x3 - i >= x1) {
    glVertex2f(x3 - i, m3 * (x3 - i) + c3);
    i++;
  }
} void
display() {
  makeTriangle(100, 100, 175.0, 175.0, 250.0,
  100.0); makeTriangle(-250, 100, -175.0, 175.0, -
  100.0, 100.0); makeTriangle(100, -100, 175.0, -
  175.0, 250.0, -100.0); glEnd(); glColor3f(0, 0.0,
  0.0); glBegin(GL_LINES); glVertex2f(-500, 0);
```

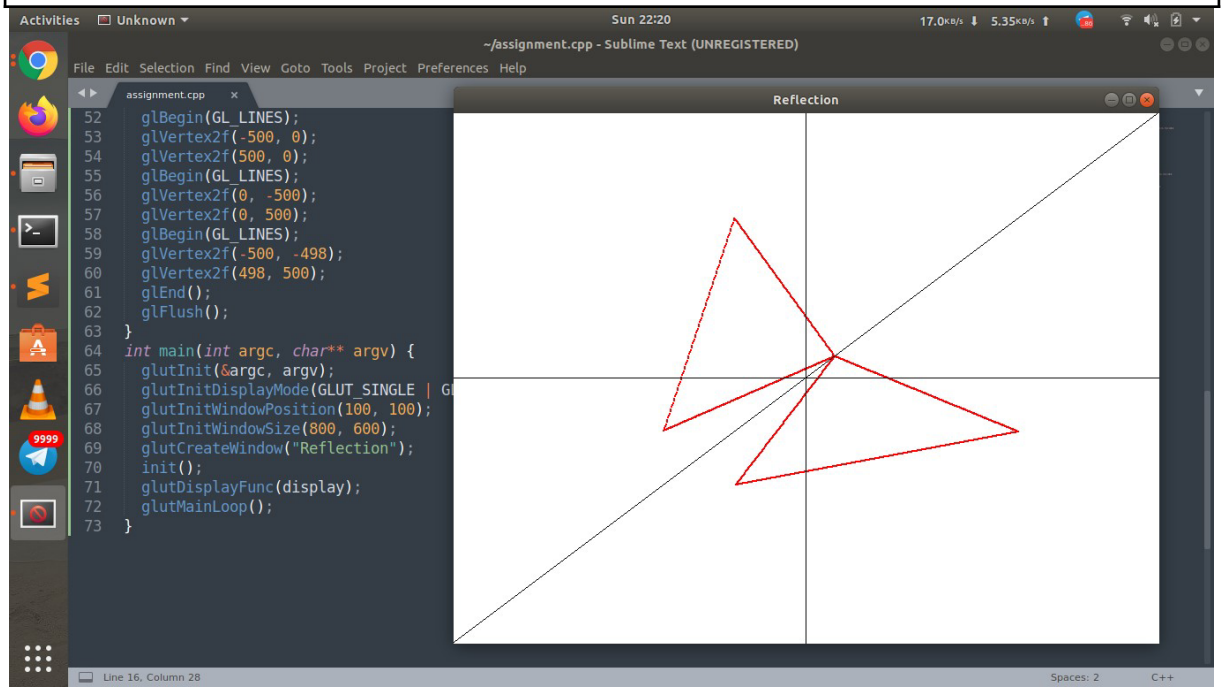
```
  glVertex2f(500, 0);
  glBegin(GL_LINES);
  glVertex2f(0, -500);
  glVertex2f(0, 500);
  glEnd();
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Reflection");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```



About line y = x + 2:

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
using namespace std;
void init() {
  glClearColor(1, 1, 1, 1.0);
  glPointSize(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); gluOrtho2D(-500,
  500.0, -500.0, 500.0);
} void makeTriangle(float x1, float y1, float x2, float y2, float x3,
float y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 0.0, 0.0);
  glBegin(GL_POINTS); float m1 =
  float(y2 - y1) / (x2 - x1); float m2
  = float(y3 - y2) / (x3 - x2); float
  m3 = float(y1 - y3) / (x1 - x3);
  float c1 = y1 - m1 *
  x1; float c2 = y2 - m2
  * x2; float c3 = y3 -
  m3 * x3; int i = 0;
  while (x1 + i <= x2) {
    glVertex2f(x1 + i, m1 * (x1 + i) +
  c1); i++; } i = 0; while (x2 + i <= x3) {
    glVertex2f(x2 + i, m2 * (x2 + i) +
  c2); i++; } i = 0; while (x3 - i >= x1) {
    glVertex2f(x3 - i, m3 * (x3 - i) + c3);
    i++;
  }
} void reflection(float x1, float y1, float x2, float y2, float x3, float
y3) {
  float new_x1 = y1 - 2; float new_y1 = x1 + 2; float new_x2 = y2 -
  2; float new_y2 = x2 + 2; float new_x3 = y3 - 2; float new_y3 =
  x3 + 2; makeTriangle(new_x1, new_y1, new_x3, new_y3,
  new_x2, new_y2);
} void
display() {
  makeTriangle(-100, -200, 40, 42, 300, -100);
```

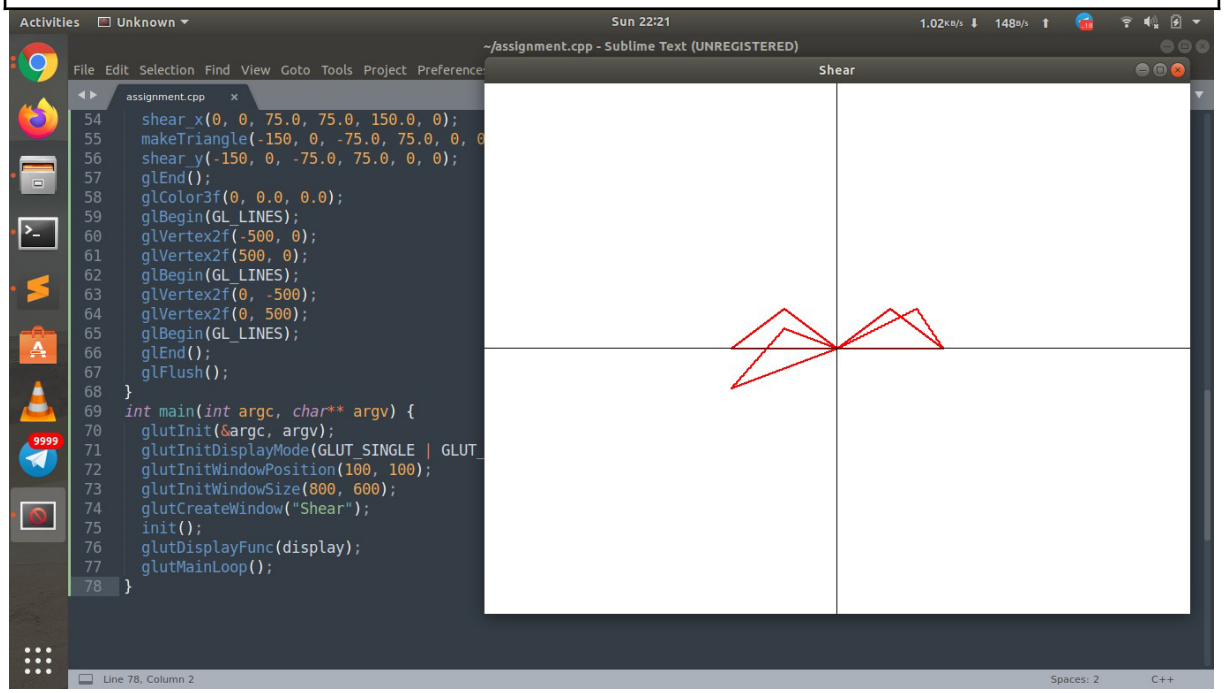
```
  reflection(-100, -200, 40, 42, 300, -
  100); glEnd(); glColor3f(0, 0.0, 0.0);
  glBegin(GL_LINES); glVertex2f(-500,
  0); glVertex2f(500, 0);
  glBegin(GL_LINES); glVertex2f(0, -
  500); glVertex2f(0, 500);
  glBegin(GL_LINES); glVertex2f(-500,
  -498); glVertex2f(498, 500); glEnd();
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Reflection");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```



(ii) Shear about x-axis and y-axis

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
using namespace std;
void init() {
  glClearColor(1, 1, 1, 1.0);
  glPointSize(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); gluOrtho2D(-500,
  500.0, -500.0, 500.0);
} void makeTriangle(float x1, float y1, float x2, float y2, float x3,
float y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 0.0, 0.0);
  glBegin(GL_POINTS); float m1 =
  float(y2 - y1) / (x2 - x1); float m2
  = float(y3 - y2) / (x3 - x2); float
  m3 = float(y1 - y3) / (x1 - x3);
  float c1 = y1 - m1 *
  x1; float c2 = y2 - m2
  * x2; float c3 = y3 -
  m3 * x3; int i = 0;
  while (x1 + i <= x2) {
    glVertex2f(x1 + i, m1 * (x1 + i) +
  c1); i++; } i = 0; while (x2 + i <= x3) {
    glVertex2f(x2 + i, m2 * (x2 + i) +
  c2); i++; } i = 0; while (x3 - i >= x1) {
    glVertex2f(x3 - i, m3 * (x3 - i) + c3);
    i++;
  }
} void shear_x(float x1, float y1, float x2, float y2, float x3, float
y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  x1 = x1 + 0.5 * y1;
  x2 = x2 + 0.5 * y2;
  x3 = x3 + 0.5 * y3;
  makeTriangle(x1, y1, x2, y2, x3, y3);
} void shear_y(float x1, float y1, float x2, float y2, float x3, float
y3) {
  glClear(GL_COLOR_BUFFER_BIT);
  y1 = y1 + 0.5 * x1;
  y2 = y2 + 0.5 * x2;
```

```
  y3 = y3 + 0.5 * x3;
  makeTriangle(x1, y1, x2, y2, x3, y3);
} void
display() {
  makeTriangle(0, 0, 75.0, 75.0, 150.0,
  0); shear_x(0, 0, 75.0, 75.0, 150.0, 0);
  makeTriangle(-150, 0, -75.0, 75.0, 0,
  0); shear_y(-150, 0, -75.0, 75.0, 0, 0);
  glEnd(); glColor3f(0, 0.0, 0.0);
  glBegin(GL_LINES); glVertex2f(-500,
  0); glVertex2f(500, 0);
  glBegin(GL_LINES); glVertex2f(0, -
  500); glVertex2f(0, 500);
  glBegin(GL_LINES);
  glEnd();
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100, 100);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Shear");
  init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```



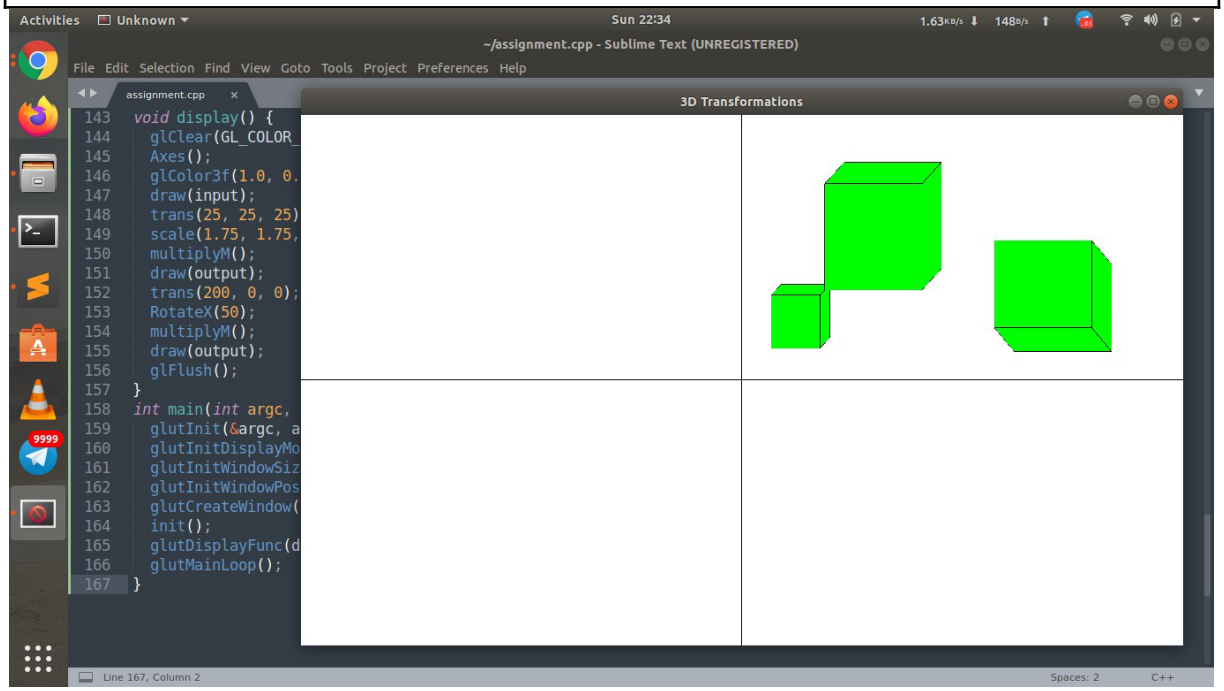
Q9: Write a program for performing the basic transformations such as translation, Scaling, Rotation for a given 3D object.

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h> using
namespace std; typedef
float Matrix4[4][4];
Matrix4 theMatrix;
static GLfloat input[8][3] = {{40, 40, -50}, {90, 40, -50}, {90, 90, -50},
                        {40, 90, -50}, {30, 30, 0},   {80, 30, 0}, {80,
                        80, 0},   {30, 80, 0}};
float output[8][3];
void init() {
  glClearColor(1.0, 1.0, 1.0, 1.0); glOrtho(-454.0,
  454.0, -250.0, 250.0, -250.0, 250.0);
  glEnable(GL_DEPTH_TEST);
}
void setIdentityM(Matrix4 m) {
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++) m[i][j] = (i == j);
} void
Axes(void) {
  glColor3f(0.0, 0.0, 0.0);
  glBegin(GL_LINES);
  glVertex2s(-1000, 0);
  glVertex2s(1000, 0);
  glEnd();
  glBegin(GL_LINES);
glVertex2s(0, -1000);
glVertex2s(0, 1000);
glEnd(); }
void draw(float a[8][3]) {
  glBegin(GL_QUADS);
  glColor3f(0, 1, 0);
  glVertex3fv(a[0]);
  glVertex3fv(a[1]);
  glVertex3fv(a[2]);
  glVertex3fv(a[3]);
  glColor3f(0, 1, 0);
  glVertex3fv(a[0]);
  glVertex3fv(a[1]);
  glVertex3fv(a[5]);
  glVertex3fv(a[4]);
  glColor3f(0, 1, 0);
  glVertex3fv(a[0]);
  glVertex3fv(a[4]);
  glVertex3fv(a[7]);
```

```
glVertex3fv(a[3]);
glColor3f(0, 1, 0);
glVertex3fv(a[1]);
glVertex3fv(a[2]);
glVertex3fv(a[6]);
glVertex3fv(a[5]);
glColor3f(0, 1, 0);
glVertex3fv(a[2]);
glVertex3fv(a[3]);
glVertex3fv(a[7]);
glVertex3fv(a[6]);
glColor3f(0, 1, 0);
glVertex3fv(a[4]);
glVertex3fv(a[5]);
glVertex3fv(a[6]);
glVertex3fv(a[7]); glEnd();
glColor3f(0, 0, 0);
glBegin(GL_LINE_LOOP);
glVertex3fv(a[0]);
glVertex3fv(a[1]);
glVertex3fv(a[1]);
glVertex3fv(a[2]);
glVertex3fv(a[2]);
glVertex3fv(a[3]); glEnd();
glColor3f(0, 0, 0);
glBegin(GL_LINE_STRIP);
glVertex3fv(a[0]);
glVertex3fv(a[1]);
glVertex3fv(a[5]);
glVertex3fv(a[4]);
glVertex3fv(a[0]); glEnd();
glColor3f(0, 0, 0);
glBegin(GL_LINE_STRIP);
glVertex3fv(a[0]);
glVertex3fv(a[4]);
glVertex3fv(a[7]);
glVertex3fv(a[3]);
glVertex3fv(a[0]); glEnd();
glColor3f(0, 0, 0);
glBegin(GL_LINE_STRIP);
glVertex3fv(a[1]);
glVertex3fv(a[2]);
glVertex3fv(a[6]);
glVertex3fv(a[5]);
glVertex3fv(a[1]);
```

```
  glEnd(); glColor3f(0, 0, 0);
glBegin(GL_LINE_STRIP);
glVertex3fv(a[2]);
glVertex3fv(a[3]);
glVertex3fv(a[7]);
glVertex3fv(a[6]);
glVertex3fv(a[2]); glEnd();
glColor3f(0, 0, 0);
glBegin(GL_LINE_STRIP);
glVertex3fv(a[4]);
glVertex3fv(a[5]);
glVertex3fv(a[6]);
glVertex3fv(a[7]);
glVertex3fv(a[4]); glEnd(); }
void RotateX(float angle) // Parallel to x
{ angle = angle * 3.142 /
  180; theMatrix[1][1] =
  cos(angle); theMatrix[1][2]
  = -sin(angle);
  theMatrix[2][1] =
  sin(angle); theMatrix[2][2]
  = cos(angle);
} void scale(int sx, int sy, int
sz) {
  theMatrix[0][0] = sx;
  theMatrix[1][1] = sy;
  theMatrix[2][2] = sz;
} void
multiplyM() {
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 3; j++) {
      for (int k = 0; k < 3; k++) {
        output[i][j] = output[i][j] + input[i][k] * theMatrix[k][j]; }
    }
  }
}
void trans(int tx, int ty, int tz)
  { for (int i = 0; i < 8; i++) {
    output[i][0] = input[i][0] + tx;
    output[i][1] = input[i][1] + ty;
    output[i][2] = input[i][2] + tz;
  }
} void
display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
  Axes();
  glColor3f(1.0, 0.0,
  0.0); draw(input);
  trans(25, 25, 25);
  scale(1.75, 1.75,
  1.75); multiplyM();
  draw(output);
  trans(200, 0, 0);
  RotateX(50);
  multiplyM();
  draw(output);
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(1000, 600); glutInitWindowPosition(0, 0);
  glutCreateWindow("3D Transformations"); init();
  glutDisplayFunc(display);
  glutMainLoop();
}
```

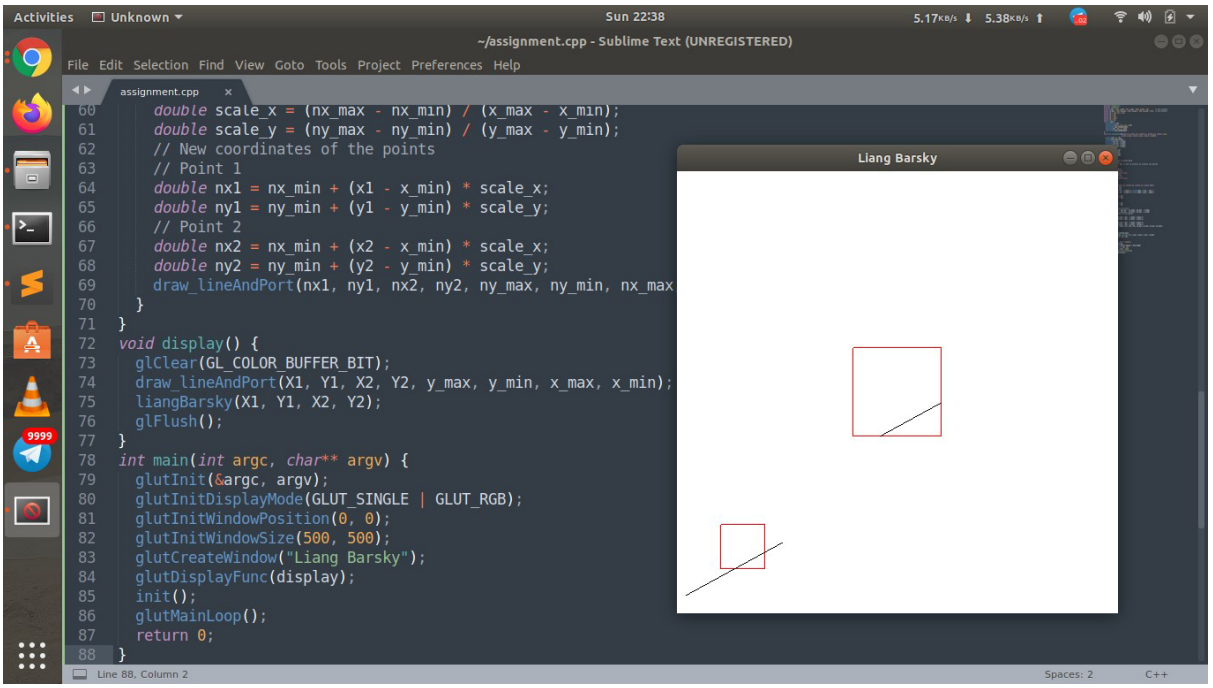


Q10: Write a program to clip a line using Liang Barsky Algorithm and Cohen Sutherland

Liang Barsky Algorithm

```
#include <GL/glut.h>
#include <stdlib.h>
double y_max = 100, y_min = 50, x_max = 100, x_min = 50;     // Old viewport
double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200; // New
ViewPort double t1 = 0.0, t2 = 1.0; double X1 = 10; double Y1 = 20; double X2 =
120; double Y2 = 80;
void init() {
  glLoadIdentity();
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glMatrixMode(GL_PROJECTION)
  ; gluOrtho2D(0, 500, 0, 500);
  glMatrixMode(GL_MODELVIEW);
}
void draw_lineAndPort(double x1, double y1, double x2, double y2, double y_max,
                double y_min, double x_max, double x_min) {
  glColor3d(1, 0, 0);
  glBegin(GL_LINE_LOOP);
  glVertex2d(x_min, y_min);
  glVertex2d(x_max, y_min);
  glVertex2d(x_max, y_max);
  glVertex2d(x_min, y_max);
  glEnd(); glColor3d(0, 0, 0);
glBegin(GL_LINES);
glVertex2d(x1, y1);
glVertex2d(x2, y2); glEnd(); }
bool cliptest(double p, double
q) {
  double t = q / p; if (p == 0 && q < 0) // Line is parallel to
  viewport and outside
  { return
    false;
  } else if (p < 0) { if (t >
    t1) t1 = t; if (t > t2)
    return false;
  } else if (p > 0) { if (t <
    t2) t2 = t; if (t < t1)
    return false;
  }
  return true;
}
void liangBarsky(double x1, double y1, double x2, double y2) {
```

```
  double dx = x2 - x1;
  double dy = y2 - y1;
  if (cliptest(-dx, x1 - x_min) && cliptest(dx, x_max - x1) && cliptest(-
     dy, y1 - y_min) && cliptest(dy, y_max - y1)) {
   if (t2 < 1) {
    x2 = x1 + t2 * dx;
    y2 = y1 + t2 * dy;
   } if (t1 >
   0) {
    x1 = x1 + t1 * dx;
    y1 = y1 + t1 * dy;
   }
   // Scaling to new View port
   double scale_x = (nx_max - nx_min) / (x_max - x_min);
   double scale_y = (ny_max - ny_min) / (y_max - y_min);
   // New coordinates of the points
   // Point 1
   double nx1 = nx_min + (x1 - x_min) * scale_x;
   double ny1 = ny_min + (y1 - y_min) * scale_y;
   // Point 2
   double nx2 = nx_min + (x2 - x_min) * scale_x; double ny2 = ny_min +
   (y2 - y_min) * scale_y; draw_lineAndPort(nx1, ny1, nx2, ny2, ny_max,
   ny_min, nx_max, nx_min);
  }
} void
display() {
  glClear(GL_COLOR_BUFFER_BIT);   draw_lineAndPort(X1,   Y1,
  X2, Y2, y_max, y_min, x_max, x_min); liangBarsky(X1, Y1, X2,
  Y2);
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(0, 0);
  glutInitWindowSize(500, 500);
  glutCreateWindow("Liang Barsky");
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;
}
```

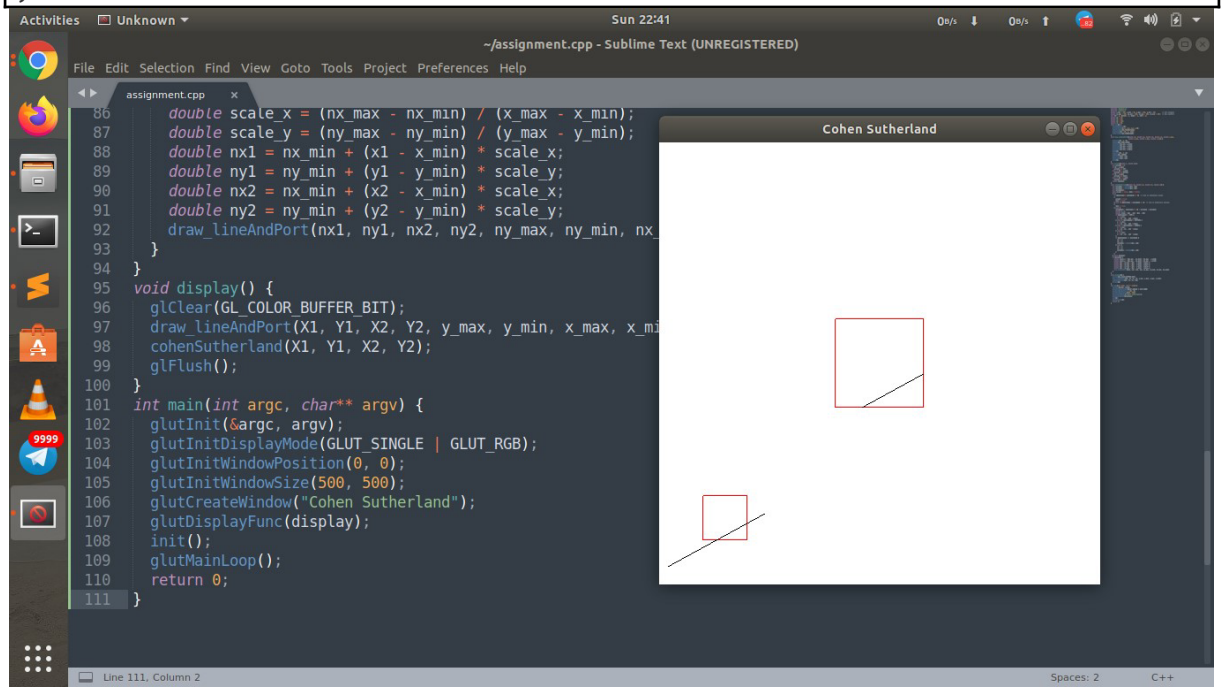

Activities
Unknown
Sun 22:38
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
double scale_x = (nx_max - nx_min) / (x_max - x_min);
double scale_y = (ny_max - ny_min) / (y_max - y_min);
// New coordinates of the points
// Point 1
double nx1 = nx_min + (x1 - x_min) * scale_x;
double ny1 = ny_min + (y1 - y_min) * scale_y;
// Point 2
double nx2 = nx_min + (x2 - x_min) * scale_x;
double ny2 = ny_min + (y2 - y_min) * scale_y;
draw_lineAndPort(nx1, ny1, nx2, ny2, ny_max, ny_min, nx_max
}
}
void display() {
glClear(GL_COLOR_BUFFER_BIT);
draw_lineAndPort(X1, Y1, X2, Y2, y_max, y_min, x_max, x_min);
liangBarsky(X1, Y1, X2, Y2);
glFlush();
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowPosition(0, 0);
glutInitWindowSize(500, 500);
glutCreateWindow("Liang Barsky");
glutDisplayFunc(display);
init();
glutMainLoop();
return 0;
}
Liang Barsky
Line 88, Column 2
Spaces: 2
C++

Cohen Sutherland

```
#include <GL/glut.h>
#include <stdlib.h>
double y_max = 100, y_min = 50, x_max = 100, x_min = 50;     // Old viewport
double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200; // New
ViewPort int TOP = 8, BOTTOM = 4, RIGHT = 2, LEFT = 1; double X1 = 10;
double Y1 = 20; double X2 = 120; double Y2 = 80;
void init() {
  glLoadIdentity();
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glMatrixMode(GL_PROJECTION)
  ; gluOrtho2D(0, 500, 0, 500);
  glMatrixMode(GL_MODELVIEW);
}
void draw_lineAndPort(double x1, double y1, double x2, double y2, double y_max,
                double y_min, double x_max, double x_min) {
  glColor3d(1, 0, 0);
  glBegin(GL_LINE_LOOP);
  glVertex2d(x_min, y_min);
  glVertex2d(x_max, y_min);
  glVertex2d(x_max, y_max);
  glVertex2d(x_min, y_max);
  glEnd(); glColor3d(0, 0, 0);
glBegin(GL_LINES);
glVertex2d(x1, y1);
glVertex2d(x2, y2); glEnd(); }
int outcode(double x, double
y) {
  int outcode = 0;
  if (y > y_max)
    outcode |= TOP;
  else if (y < y_min)
    outcode |= BOTTOM;
  if (x > x_max)
    outcode |= RIGHT;
  else if (x < x_min)
    outcode |= LEFT;
  return outcode;
} void cohenSutherland(double x1, double y1, double x2,
double y2) {
  int outcode1 = outcode(x1, y1);
  int outcode2 = outcode(x2, y2);
  int outcodeOut;
  bool accept = false, done = false;
```

```
 do {
  if ((outcode1 | outcode2) == 0) // line is completely inside
  { accept =
    true; done =
    true;
  } else if ((outcode1 & outcode2) != 0) // line is completely outside
  { done =
    true;
  } else { outcodeOut = (outcode1 != 0) ? outcode1
     : outcode2; double x, y; double slope = (y2 - y1)
    / (x2 - x1); if (outcodeOut & TOP) { y = y_max;
     x = x1 + (y - y1) / slope;
    } else if (outcodeOut & BOTTOM)
     { y = y_min;
     x = x1 + (y - y1) / slope;
    } else if (outcodeOut & RIGHT)
     { x = x_max;
     y = y1 + (x - x1) * slope;
    } else { x =
     x_min;
     y = y1 + (x - x1) * slope;
    } if (outcodeOut ==
    outcode1) {
     x1 = x;
     y1 = y;
     outcode1 = outcode(x1, y1);
    } else
     { x2 = x;
     y2 = y;
     outcode2 = outcode(x2, y2);
    }
  }
 } while (!done);
 if (accept) {
  double scale_x = (nx_max - nx_min) / (x_max - x_min); double
  scale_y = (ny_max - ny_min) / (y_max - y_min); double nx1 = nx_min
  + (x1 - x_min) * scale_x; double ny1 = ny_min + (y1 - y_min) * scale_y;
  double nx2 = nx_min + (x2 - x_min) * scale_x; double ny2 = ny_min +
  (y2 - y_min) * scale_y; draw_lineAndPort(nx1, ny1, nx2, ny2, ny_max,
  ny_min, nx_max, nx_min);
 }
} void
display() {
 glClear(GL_COLOR_BUFFER_BIT);
 draw_lineAndPort(X1, Y1, X2, Y2, y_max, y_min, x_max, x_min);
```

```
  cohenSutherland(X1, Y1, X2, Y2);
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(0, 0);
  glutInitWindowSize(500, 500);
  glutCreateWindow("Cohen Sutherland");
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;
}
```

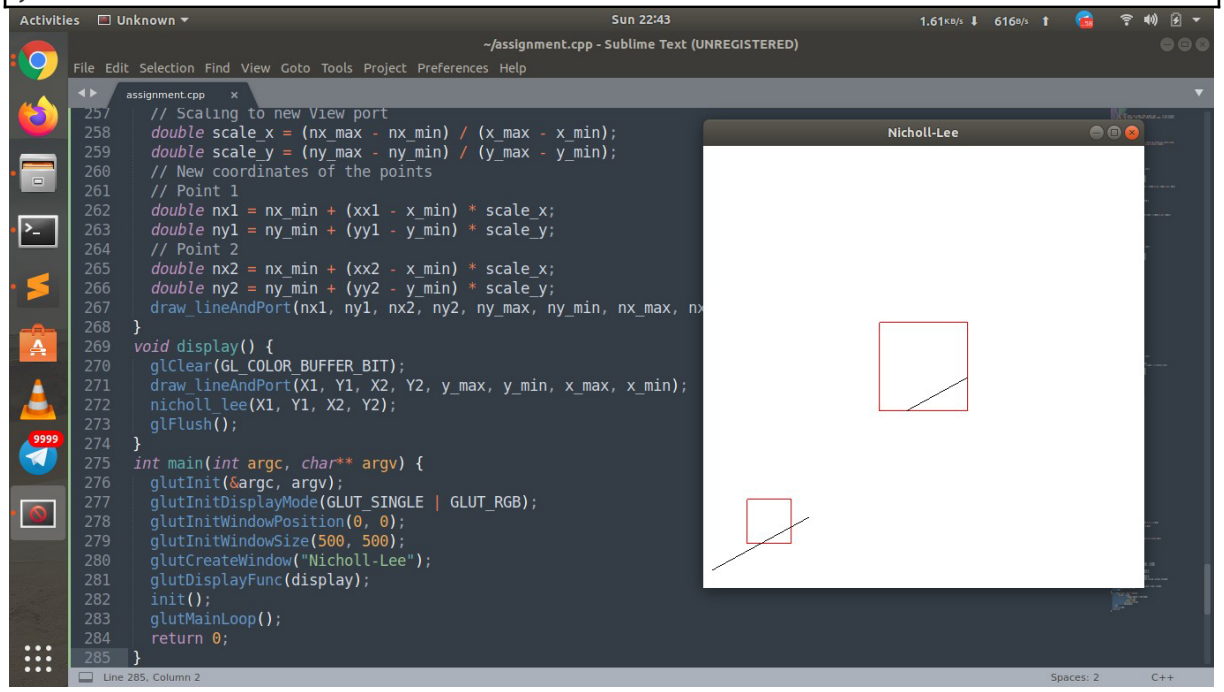


Q11: Write a program to clip a line using Nicholl-Lee-Nicholl Line clipping

```
#include <GL/glut.h>
#include <stdlib.h>
double y_max = 100, y_min = 50, x_max = 100, x_min = 50;     // Old viewport
double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200; // New
ViewPort int TOP = 8, BOTTOM = 4, RIGHT = 2, LEFT = 1; double X1 = 10;
double Y1 = 20; double X2 = 120; double Y2 = 80;
void init() {
  glLoadIdentity();
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glMatrixMode(GL_PROJECTION)
  ; gluOrtho2D(0, 500, 0, 500);
  glMatrixMode(GL_MODELVIEW);
}
void draw_lineAndPort(double x1, double y1, double x2, double y2, double y_max,
                      double y_min, double x_max, double x_min) {
  glColor3d(1, 0, 0);
  glBegin(GL_LINE_LOOP);
  glVertex2d(x_min, y_min);
  glVertex2d(x_max, y_min);
  glVertex2d(x_max, y_max);
  glVertex2d(x_min, y_max);
  glEnd(); glColor3d(0, 0, 0);
glBegin(GL_LINES);
glVertex2d(x1, y1);
glVertex2d(x2, y2); glEnd(); }
int outcode(double x, double
y) {
  int outcode = 0;
  if (y > y_max)
    outcode |= TOP;
  else if (y < y_min)
    outcode |= BOTTOM;
  if (x > x_max)
    outcode |= RIGHT;
  else if (x < x_min)
    outcode |= LEFT;
  return outcode;
} void cohenSutherland(double x1, double y1, double x2,
double y2) {
  int outcode1 = outcode(x1, y1);
  int outcode2 = outcode(x2, y2);
  int outcodeOut;
  bool accept = false, done = false;
```

```
 do {
  if ((outcode1 | outcode2) == 0) // line is completely inside
  { accept =
    true; done =
    true;
  } else if ((outcode1 & outcode2) != 0) // line is completely outside
  { done =
    true;
  } else { outcodeOut = (outcode1 != 0) ? outcode1
     : outcode2; double x, y; double slope = (y2 - y1)
    / (x2 - x1); if (outcodeOut & TOP) { y = y_max;
     x = x1 + (y - y1) / slope;
    } else if (outcodeOut & BOTTOM)
     { y = y_min;
     x = x1 + (y - y1) / slope;
    } else if (outcodeOut & RIGHT)
     { x = x_max;
     y = y1 + (x - x1) * slope;
    } else { x =
     x_min;
     y = y1 + (x - x1) * slope;
    } if (outcodeOut ==
    outcode1) {
     x1 = x;
     y1 = y;
     outcode1 = outcode(x1, y1);
    } else
     { x2 = x;
     y2 = y;
     outcode2 = outcode(x2, y2);
    }
  }
 } while (!done);
 if (accept) {
  double scale_x = (nx_max - nx_min) / (x_max - x_min); double
  scale_y = (ny_max - ny_min) / (y_max - y_min); double nx1 = nx_min
  + (x1 - x_min) * scale_x; double ny1 = ny_min + (y1 - y_min) * scale_y;
  double nx2 = nx_min + (x2 - x_min) * scale_x; double ny2 = ny_min +
  (y2 - y_min) * scale_y; draw_lineAndPort(nx1, ny1, nx2, ny2, ny_max,
  ny_min, nx_max, nx_min);
 }
} void
display() {
 glClear(GL_COLOR_BUFFER_BIT);
 draw_lineAndPort(X1, Y1, X2, Y2, y_max, y_min, x_max, x_min);
```

```
  cohenSutherland(X1, Y1, X2, Y2);
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(0, 0);
  glutInitWindowSize(500, 500);
  glutCreateWindow("Cohen Sutherland");
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;
}
```



Q12: Write a program to clip a polygon using Sutherland Hodgeman and Weiler Atherton algorithm

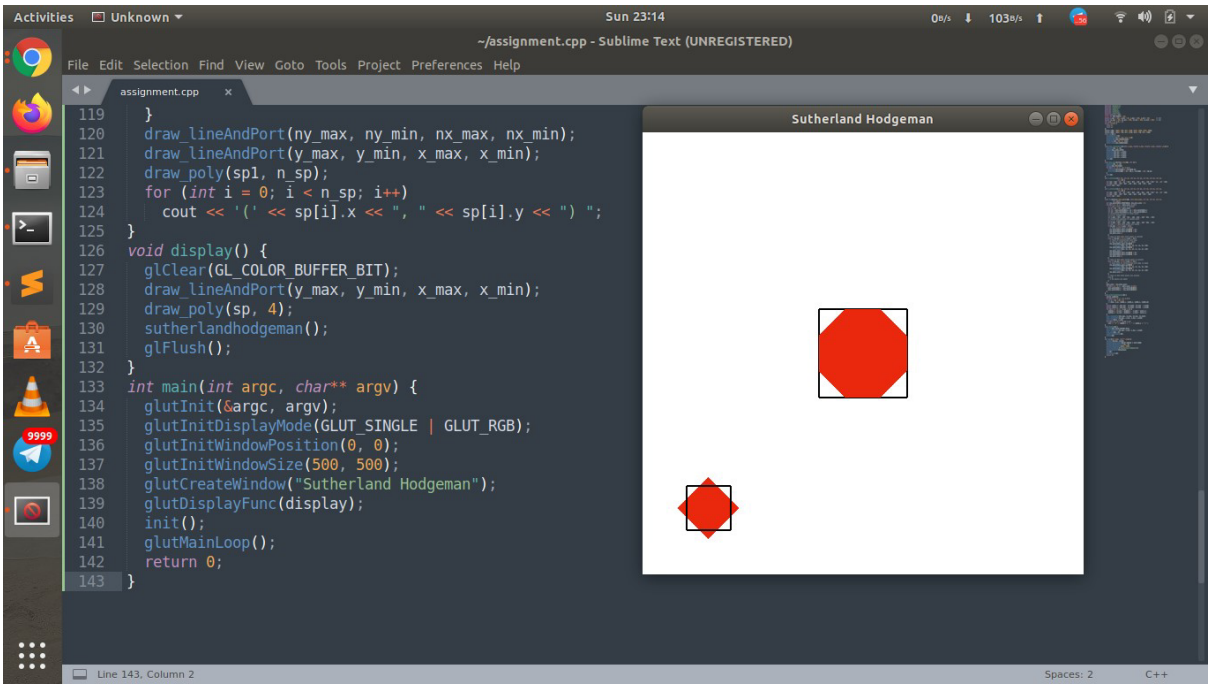
Sutherland Hodgeman:

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
#include <cstdlib>
#include <iostream> using namespace std; const int MAX_POINTS =
20; double y_max = 100, y_min = 50, x_max = 100, x_min = 50;
// Old double ny_max = 300, ny_min = 200, nx_max = 300, nx_min =
200; // New int n_sp = 4, n_cw = 4;
struct vertex {
  float x;
float y; };
vertex cw[] = {{50, 50}, {50, 100}, {100, 100}, {100, 50}};
vertex sp[] = {{40, 75}, {75, 110}, {110, 75}, {75, 40}}; void
init() {
  glLoadIdentity();
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glMatrixMode(GL_PROJECTION)
  ; gluOrtho2D(0, 500, 0, 500);
  glMatrixMode(GL_MODELVIEW);
} void draw_lineAndPort(double y_max, double y_min, double x_max, double
x_min) {
  glColor3d(0, 0, 0);
  glBegin(GL_LINE_LOOP);
  glVertex2d(x_min, y_min);
  glVertex2d(x_max, y_min);
  glVertex2d(x_max, y_max);
  glVertex2d(x_min, y_max);
  glEnd(); } void draw_poly(vertex
vlist[], int n) {
  glColor3d(1, 0, 0);
  glBegin(GL_POLYGON);
  for (int i = 0; i < n; i++) {
    glVertex2d(vlist[i].x, vlist[i].y);
    glVertex2d(vlist[(i + 1) % n].x, vlist[(i + 1) %
    n].y);
  }
glEnd(); }
int x_intersect(int x1, int y1, int x2, int y2, int x3, int y3, int x4,
                int y4) {
  int num = (x1 * y2 - y1 * x2) * (x3 - x4) - (x1 - x2) * (x3 * y4 - y3 * x4);
  int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
```

```
  return num / den;
}
int y_intersect(int x1, int y1, int x2, int y2, int x3, int y3, int x4,
                int y4) {
  int num = (x1 * y2 - y1 * x2) * (y3 - y4) - (y1 - y2) * (x3 * y4 - y3 * x4);
  int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
  return num / den;
}
void clip(vertex poly_points[], int& poly_size, int x1, int y1, int x2,
          int y2) {
  int new_points[MAX_POINTS][2], new_poly_size = 0;
  for (int i = 0; i < poly_size; i++) {
    // i and k form a line in
    polygon int k = (i + 1) %
    poly_size;
    int ix = poly_points[i].x, iy = poly_points[i].y;
    int kx = poly_points[k].x, ky =
    poly_points[k].y;
    // Calculating position of first point
    // w.r.t. clipper line
    int i_pos = (x2 - x1) * (iy - y1) - (y2 - y1) * (ix - x1);
    // Calculating position of second point
    // w.r.t. clipper line
    int k_pos = (x2 - x1) * (ky - y1) - (y2 - y1) * (kx - x1);
    // Case 1 : When both points are
    inside if (i_pos < 0 && k_pos < 0) {
      // Only second point is added
      new_points[new_poly_size][0] = kx;
      new_points[new_poly_size][1] = ky;
      new_poly_size++;
    }
    // Case 2: When only first point is
    outside else if (i_pos >= 0 && k_pos < 0)
    {
      // Point of intersection with
      edge // and the second point
      is added
      new_points[new_poly_size][0] =
          x_intersect(x1, y1, x2, y2, ix, iy, kx, ky);
      new_points[new_poly_size][1] =
          y_intersect(x1, y1, x2, y2, ix, iy, kx, ky);
      new_poly_size++;
      new_points[new_poly_size][0] = kx;
      new_points[new_poly_size][1] = ky;
      new_poly_size++;
    }
    // Case 3: When only second point is
```

```
outside else if (i_pos < 0 && k_pos >= 0) {
 // Only point of intersection with edge is
 added new_points[new_poly_size][0] =
 x_intersect(x1, y1, x2, y2, ix, iy, kx, ky);
 new_points[new_poly_size][1] =
    y_intersect(x1, y1, x2, y2, ix, iy, kx, ky);
```

```
      new_poly_size++;
    }
    // Case 4: When both points are
    outside else {
      // No points are added
    }
  } poly_size = new_poly_size;
  for (int i = 0; i < poly_size; i++) {
    poly_points[i].x = new_points[i][0];
  poly_points[i].y = new_points[i][1]; }
} void
sutherlandhodgeman() {
  vertex sp1[20]; for (int i
  = 0; i < 4; i++) { int k = (i
  + 1) % 4;
    clip(sp, n_sp, cw[i].x, cw[i].y, cw[k].x, cw[k].y);
  } double scale_x = (nx_max - nx_min) / (x_max -
  x_min); double scale_y = (ny_max - ny_min) /
  (y_max - y_min); for (int i = 0; i < n_sp; i++) {
    sp1[i].x = nx_min + (sp[i].x - x_min) * scale_x;
    sp1[i].y = ny_min + (sp[i].y - y_min) * scale_y;
  } draw_lineAndPort(ny_max, ny_min, nx_max,
  nx_min); draw_lineAndPort(y_max, y_min, x_max,
  x_min); draw_poly(sp1, n_sp); for (int i = 0; i < n_sp;
  i++)
    cout << '(' << sp[i].x << ", " << sp[i].y << ") ";
} void
display() {
  glClear(GL_COLOR_BUFFER_BIT);
  draw_lineAndPort(y_max, y_min, x_max,
  x_min); draw_poly(sp, 4);
  sutherlandhodgeman();
  glFlush();
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(0, 0);
  glutInitWindowSize(500, 500);
  glutCreateWindow("Sutherland Hodgeman");
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;
}
```


Activities
Unknown
Sun 23:14
~/assignment.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
assignment.cpp
}
draw_lineAndPort(ny_max, ny_min, nx_max, nx_min);
draw_lineAndPort(y_max, y_min, x_max, x_min);
draw_poly(sp1, n_sp);
for (int i = 0; i < n_sp; i++)
cout << '(' << sp[i].x << ", " << sp[i].y << ") ";
}
void display() {
glClear(GL_COLOR_BUFFER_BIT);
draw_lineAndPort(y_max, y_min, x_max, x_min);
draw_poly(sp, 4);
sutherlandhodgeman();
glFlush();
}
int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowPosition(0, 0);
glutInitWindowSize(500, 500);
glutCreateWindow("Sutherland Hodgeman");
glutDisplayFunc(display);
init();
glutMainLoop();
return 0;
}
Sutherland Hodgeman
Line 143, Column 2
Spaces: 2
C++

Weiler Atherton algorithm

```
#include <iostream>
#include <cstdlib>
#include <vector>
#include <list>
#include <GL/glut.h>
#define Size 500
using namespace
std; typedef float
Color[3]; struct Point
{
  int x, y; };
typedef struct IntersectionPoint {
  int pointFlag; int
  index0, index1;
  Point p; bool
  inFlag; int dis;
} IP;
double y_max = 100, y_min = 50, x_max = 100, x_min = 50;     // Old viewport
double ny_max = 300, ny_min = 200, nx_max = 300, nx_min = 200; // New
ViewPort double scale_x = (nx_max - nx_min) / (x_max - x_min); double
scale_y = (ny_max - ny_min) / (y_max - y_min); class Pg {
 public:
  vector<Point> pts;
  Pg(void);
  void drawPgLine(Color c);
  void drawpoly();
};
Pg::Pg(void) {} void
Pg::drawPgLine(Color c) {
  glColor3fv(c);
  glLineWidth(2.0); glBegin(GL_LINE_LOOP); int size
= pts.size(); for (int i = 0; i < size; i++)
glVertex2i(pts[i].x, pts[i].y); glEnd(); } void
Pg::drawpoly() {
  glColor3d(1, 1, 102.0 / 255);
  glBegin(GL_POLYGON);
  int size = pts.size(); for
  (int i = 0; i < size; i++) {
    glVertex2d(pts[i].x, pts[i].y); glVertex2d(pts[(i + 1)
     % size].x, pts[(i + 1) % size].y);
  }
  glEnd();
}
```

```
bool isPointInsidePg(Point p, Pg& py) {
int cnt = 0, size = py.pts.size();
for (int i = 0; i < size; i++) {
  Point p1 = py.pts[i];
  Point p2 = py.pts[(i + 1) % size];
  if (p1.y == p2.y) continue; if (p.y
  < min(p1.y, p2.y)) continue;
  if (p.y >= max(p1.y, p2.y)) continue;
  double x =
     (double)(p.y - p1.y) * (double)(p2.x - p1.x) / (double)(p2.y - p1.y) +
     p1.x;
  if (x > p.x) cnt++;
}
return (cnt % 2 == 1);
} int cross(Point& p0, Point& p1, Point&
p2) {
return ((p2.x - p0.x) * (p1.y - p0.y) - (p1.x - p0.x) * (p2.y - p0.y));
}
bool onSegment(Point& p0, Point& p1, Point& p2)
{ int minx = min(p0.x, p1.x), maxx = max(p0.x,
p1.x);
int miny = min(p0.y, p1.y), maxy = max(p0.y, p1.y); if (p2.x >= minx && p2.x <=
maxx && p2.y >= miny && p2.y <= maxy) return true; return false;
} bool segmentsIntersect(Point& p1, Point& p2, Point& p3,
Point& p4) {
int d1 = cross(p3, p4, p1);
int d2 = cross(p3, p4, p2);
int d3 = cross(p1, p2, p3);
int d4 = cross(p1, p2, p4);
if (((d1 > 0 && d2 < 0) || (d1 < 0 && d2 > 0)) &&
    ((d3 > 0 && d4 < 0) || (d3 < 0 && d4 > 0)))
  return true; if (d1 == 0 && onSegment(p3, p4,
p1)) return true; if (d2 == 0 && onSegment(p3,
p4, p2)) return true; if (d3 == 0 &&
onSegment(p1, p2, p3)) return true; if (d4 == 0
&& onSegment(p1, p2, p4)) return true; return
false;
}
Point getintersectPoint(Point p1, Point p2, Point p3, Point p4) {
Point p;
int b1 = (p2.y - p1.y) * p1.x + (p1.x - p2.x) * p1.y; int b2 =
(p4.y - p3.y) * p3.x + (p3.x - p4.x) * p3.y; int D = (p2.x - p1.x)
* (p4.y - p3.y) - (p4.x - p3.x) * (p2.y - p1.y); int D1 = b2 *
(p2.x - p1.x) - b1 * (p4.x - p3.x); int D2 = b2 * (p2.y - p1.y) -
```

```
b1 * (p4.y - p3.y); p.x = D1 / D;
p.y = D2 / D;
return p;
}
void generateIntersectPoints(Pg& pyclip, Pg& py, list<IP>& iplist) {
int clipSize = pyclip.pts.size(), pySize = py.pts.size();
for (int i = 0; i < clipSize; i++) {
  Point p1 = pyclip.pts[i];
  Point p2 = pyclip.pts[(i + 1) % clipSize];
  for (int j = 0; j < pySize; j++) {
    Point p3 = py.pts[j];
    Point p4 = py.pts[(j + 1) % pySize]; if
    (segmentsIntersect(p1, p2, p3, p4))
    { IP ip; ip.index0 = j; ip.index1 = i; ip.p
    = getintersectPoint(p1, p2, p3, p4);
    iplist.push_back(ip);
    }
  }
}
} int getDistance(Point& p1, Point&
p2) {
return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y);
} bool distanceComparator(IP& ip1, IP& ip2) { return ip1.dis <
ip2.dis; } void generateList(Pg& py, list<IP>& iplist, list<IP>&
comlist, int index) {
int size = py.pts.size();
list<IP>::iterator it; for (int
i = 0; i < size; i++) {
  Point p1 = py.pts[i];
  IP ip; ip.pointFlag = 0; ip.p = p1;
  comlist.push_back(ip); list<IP> oneSeg;
  for (it = iplist.begin(); it != iplist.end(); it++)
  {
     if ((index == 0 && i == it->index0) || (index == 1 && i == it->index1)) { it
     ->dis = getDistance(it->p, p1);
     it->pointFlag = 1;
     oneSeg.push_back(*it);
   }
  }
  oneSeg.sort(distanceComparator); for (it = oneSeg.begin(); it !=
oneSeg.end(); it++) comlist.push_back(*it); }
} void getPgPointInOut(list<IP>& Pglist, Pg&
pyclip) {
```

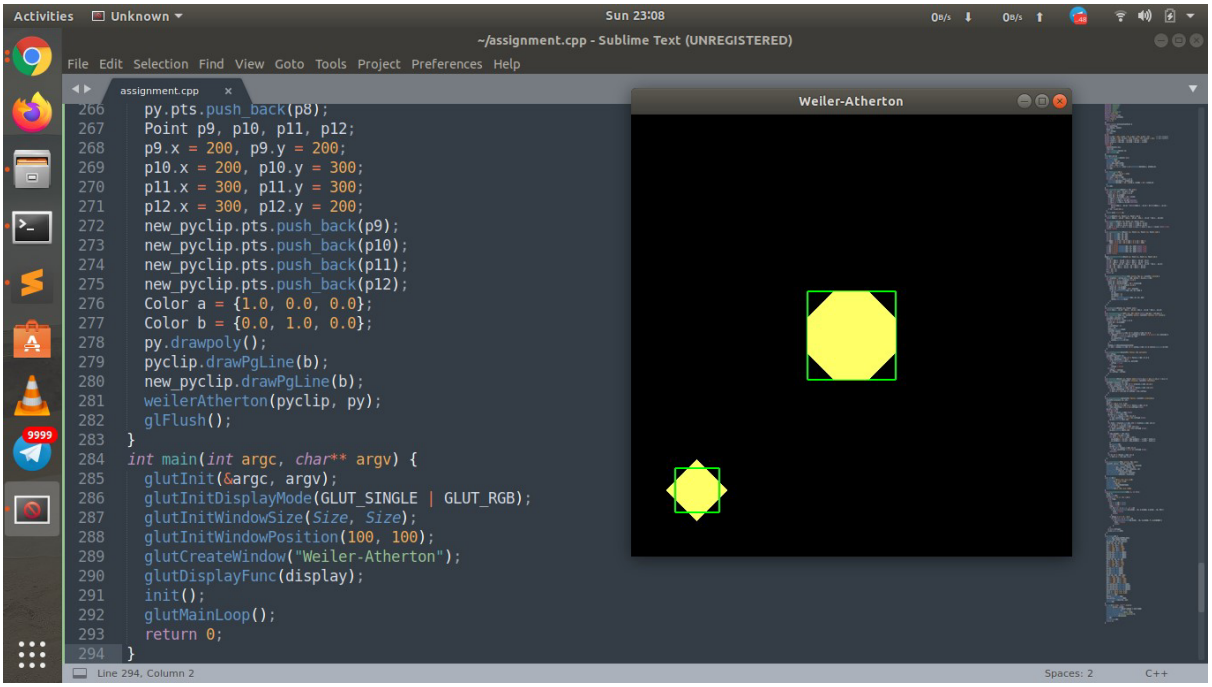
```
bool inFlag;
list<IP>::iterator it; for (it = Pglist.begin(); it
 != Pglist.end(); it++) {
  if (it->pointFlag == 0) {
    if (isPointInsidePg(it->p, pyclip))
      inFlag = true;
     else inFlag =
     false;
   } else { inFlag =
    !inFlag; it->inFlag
   = inFlag;
  }
}
}
bool operator==(Point& p1, Point& p2) { return p1.x == p2.x && p1.y == p2.y; }
void getClipPointInOut(list<IP>& cliplist, list<IP>&
Pglist) { list<IP>::iterator it, it1; for (it = cliplist.begin(); it
 != cliplist.end(); it++) { if (it->pointFlag == 0) continue;
for (it1 = Pglist.begin(); it1 != Pglist.end(); it1++) {
    if (it1->pointFlag == 0) continue; if (it->p
    == it1->p) it->inFlag = it1->inFlag;
  }
}
} void generateClipArea(list<IP>& Pglist, list<IP>&
cliplist) { list<IP>::iterator it, it1;
Pg py;
Color c = {0.0, 0.0, 1.0};
for (it = Pglist.begin(); it != Pglist.end(); it++)
  if (it->pointFlag == 1 && it->inFlag) break;
py.pts.clear();
while (true) {
  if (it == Pglist.end()) break; py.pts.push_back(it
  ->p);
  for (; it != Pglist.end(); it++) {
    if (it->pointFlag == 1 && !it->inFlag) break;
    py.pts.push_back(it->p);
  } for (it1 = cliplist.begin(); it1 != cliplist.end();
  it1++)
    if (it1->p == it->p) break;
  for (; it1 != cliplist.end(); it1++) {
    if (it1->pointFlag == 1 && it1->inFlag) break;
    py.pts.push_back(it1->p);
  }
```

```
  if (py.pts[0] == it1->p) {
    int size = py.pts.size();
    for (int i = 0; i < size; ++i) {
      py.pts[i].x = nx_min + (py.pts[i].x - x_min) * scale_x;
      py.pts[i].y = ny_min + (py.pts[i].y - y_min) * scale_y;
    } py.drawpoly();
    py.pts.clear(); for (; it !=
    Pglist.end(); it++) if (it
    ->pointFlag == 1 && it
    ->inFlag) break;
    continue; }
  for (; it != Pglist.end(); it++)
    if (it->p == it1->p) break;
}
}
 void weilerAtherton(Pg& pyclip, Pg& py)
 { list<IP> iplist, Pglist, cliplist;
 generateIntersectPoints(pyclip, py, iplist);
 generateList(py, iplist, Pglist, 0);
 generateList(pyclip, iplist, cliplist, 1);
 getPgPointInOut(Pglist, pyclip);
 getClipPointInOut(cliplist, Pglist);
 generateClipArea(Pglist, cliplist);
 }
 void init() {
 glClearColor(0.0, 0.0, 0.0, 0.0);
 glColor3f(1.0, 0.0, 0.0);
 glPointSize(1.0);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(0.0, 500, 0.0, 500);
 } void GenerateRandomSimplePg(Pg& G, int
 M) {
 Point P;
 G.pts.clear();
 for (int i = 0; i < M; ++i) {
   bool flag;
   do {
     P.x = rand() % Size;
     P.y = rand() % Size; flag
     = true; for (int j = 1; j < i -
     1; ++j)
  if (segmentsIntersect(G.pts[j - 1], G.pts[j], G.pts[i - 1], P)) {
        flag = false;
```

```
      break;
    } if (flag && i == M - 1)
   {
    for (int j = 2; j < i; ++j)
     if (segmentsIntersect(G.pts[j - 1], G.pts[j], P, G.pts[0])) {
      flag = false;
      break;
     }
   }
  } while (!flag);
  G.pts.push_back(P);
}
}
void display() {
glClear(GL_COLOR_BUFFER_BIT); glEnable(GL_POINT_SMOOTH);
  Pg pyclip, py, new_pyclip;
Point p1, p2, p3, p4; p1.x = 50,
p1.y = 50; p2.x = 50, p2.y =
100; p3.x = 100, p3.y = 100;
p4.x = 100, p4.y = 50;
pyclip.pts.push_back(p1);
pyclip.pts.push_back(p2);
pyclip.pts.push_back(p3);
pyclip.pts.push_back(p4);
Point p5, p6, p7, p8; p5.x = 40,
p5.y = 75; p6.x = 75, p6.y =
110; p7.x = 110, p7.y = 75;
p8.x = 75, p8.y = 40;
py.pts.push_back(p5);
py.pts.push_back(p6);
py.pts.push_back(p7);
py.pts.push_back(p8); Point
p9, p10, p11, p12; p9.x = 200,
p9.y = 200; p10.x = 200, p10.y
= 300; p11.x = 300, p11.y =
300; p12.x = 300, p12.y = 200;
```

```
new_pyclip.pts.push_back(p9);
new_pyclip.pts.push_back(p1
0);
new_pyclip.pts.push_back(p1
1);
new_pyclip.pts.push_back(p1
2); Color a = {1.0, 0.0, 0.0};
Color b = {0.0, 1.0, 0.0};
py.drawpoly();
pyclip.drawPgLine(b);
new_pyclip.drawPgLine(b);
weilerAtherton(pyclip, py);
glFlush(); }
int main(int argc, char** argv) {
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 glutInitWindowSize(Size, Size);
 glutInitWindowPosition(100, 100);
 glutCreateWindow("Weiler-Atherton");
 glutDisplayFunc(display);
 init();
 glutMainLoop();
 return 0;
```

```
}
```




```
  py.pts.push_back(p8);
  Point p9, p10, p11, p12;
  p9.x = 200, p9.y = 200;
  p10.x = 200, p10.y = 300;
  p11.x = 300, p11.y = 300;
  p12.x = 300, p12.y = 200;
  new_pyclip.pts.push_back(p9);
  new_pyclip.pts.push_back(p10);
  new_pyclip.pts.push_back(p11);
  new_pyclip.pts.push_back(p12);
  Color a = {1.0, 0.0, 0.0};
  Color b = {0.0, 1.0, 0.0};
  py.drawpoly();
  pyclip.drawPgLine(b);
  new_pyclip.drawPgLine(b);
  weilerAtherton(pyclip, py);
  glFlush();
}
int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(Size, Size);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Weiler-Atherton");
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;
}
```

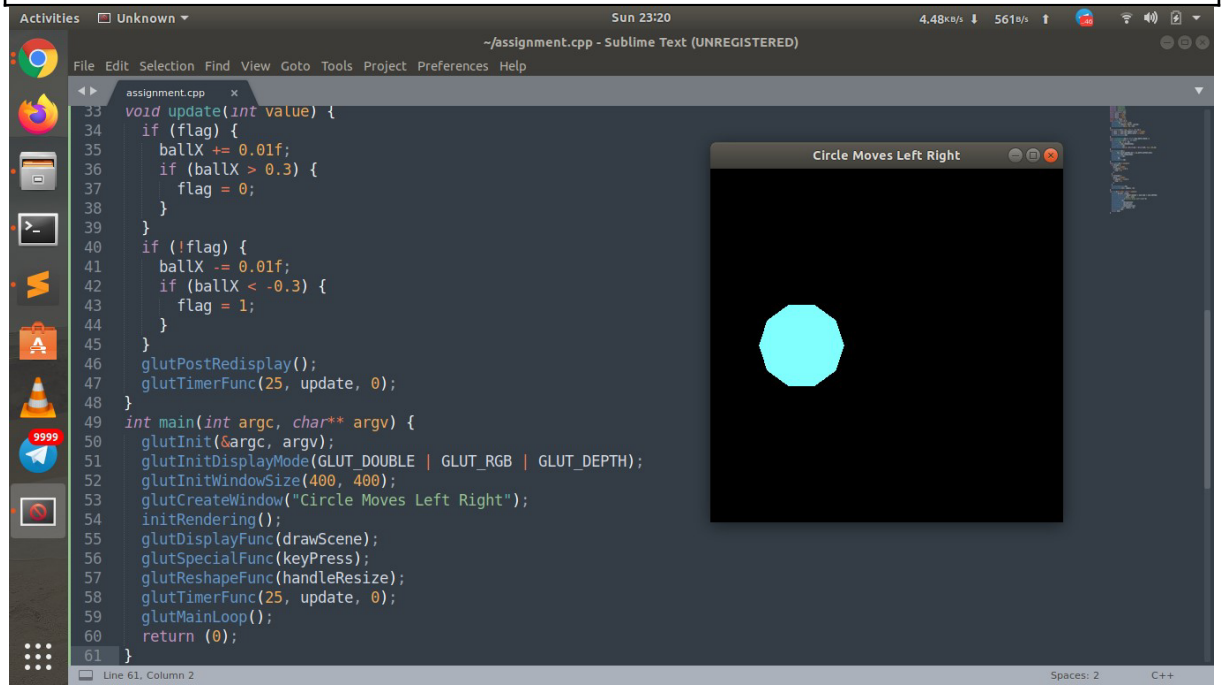
Q13: Write programs for designing following simple animations using transformations.

(i) Circle moving from left to right and vice versa

```
#include <iostream>
#include <stdlib.h>
#include
<GL/glut.h> using
namespace std;
float ballX = -0.3f;
float ballY = 0.0f;
float ballZ = -1.0f;
static int flag = 1;
void drawBall(void)
{
  glColor3f(0.5, 1.0, 1.0);
  glTranslatef(ballX, ballY, ballZ);
  glutSolidSphere(0.1, 10, 10);
} void keyPress(int key, int x, int
y) {
  if (key == GLUT_KEY_RIGHT) ballX -= 0.05f;
  if (key == GLUT_KEY_LEFT) ballX += 0.05f;
  glutPostRedisplay();
}
void initRendering() { glEnable(GL_DEPTH_TEST); }
void handleResize(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); gluPerspective(45.0, (double)w /
  (double)h, 1.0, 200.0);
} void
drawScene() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  drawBall();
  glutSwapBuffers();
} void update(int
value) {
  if (flag) {
    ballX += 0.01f;
    if (ballX > 0.3) {
      flag = 0;
    }
  }
  if (!flag) {
    ballX -= 0.01f; if
    (ballX < -0.3) {
      flag = 1;
    }
```

```
}
glutPostRedisplay();
```

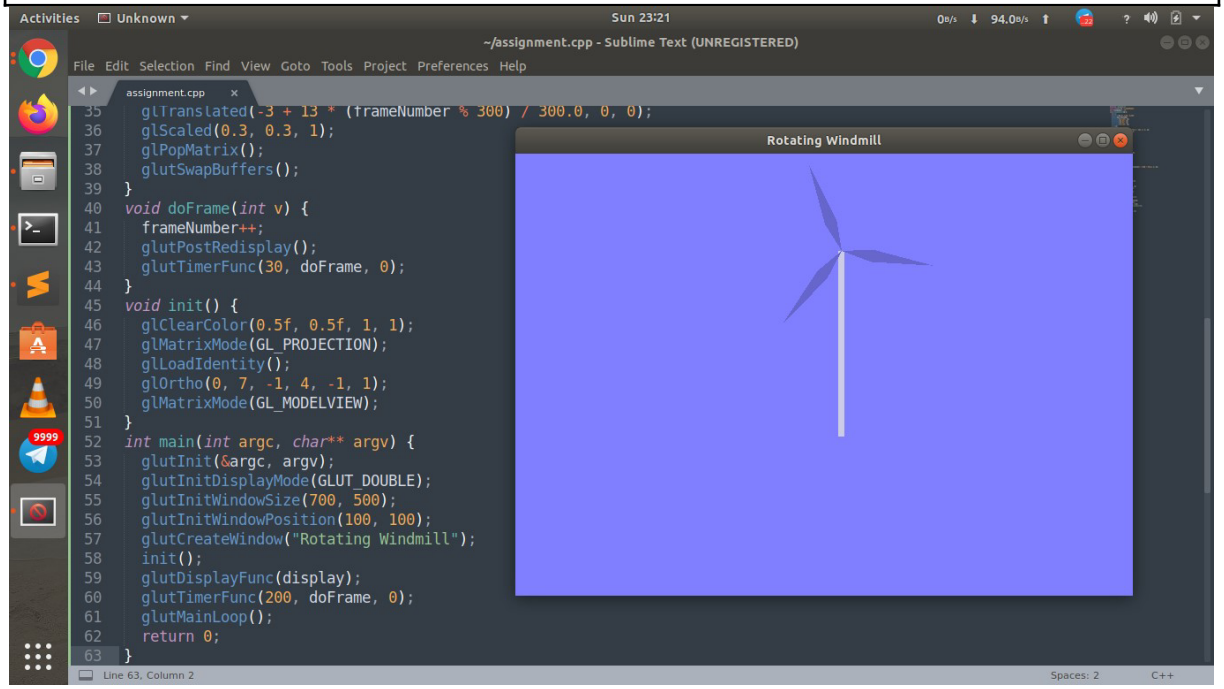
```
  glutTimerFunc(25, update, 0);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(400, 400); glutCreateWindow("Circle Moves
  Left Right"); initRendering(); glutDisplayFunc(drawScene);
  glutSpecialFunc(keyPress); glutReshapeFunc(handleResize);
  glutTimerFunc(25, update, 0); glutMainLoop(); return (0);
}
```



(ii) Wind mill rotation

```
#include <GL/glut.h> const
double PI = 3.141592654; int
frameNumber = 0;
void drawWindmill() { int
  i; glColor3f(0.8f, 0.8f,
  0.9f);
  glBegin(GL_POLYGON);
  glVertex2f(-0.05f, 0);
  glVertex2f(0.05f, 0);
  glVertex2f(0.05f, 3);
  glVertex2f(-0.05f, 3);
  glEnd();
  glTranslatef(0, 3, 0);
  glRotated(frameNumber * (180.0 / 46), 0,
  0, 1);
  glColor3f(0.4f, 0.4f, 0.8f);
  for (i = 0; i < 3; i++) {
    glRotated(120, 0, 0, 1);
    glBegin(GL_POLYGON);
    glVertex2f(0, 0);
    glVertex2f(0.5f, 0.1f);
    glVertex2f(1.5f, 0);
    glVertex2f(0.5f, -0.1f);
    glEnd();
  }
} void
display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity(); glPushMatrix(); glTranslated(3.7, 0.8,
  0); glScaled(0.7, 0.7, 1); drawWindmill();
  glPopMatrix(); glPushMatrix(); glTranslated(-3 + 13 *
  (frameNumber % 300) / 300.0, 0, 0); glScaled(0.3, 0.3,
  1); glPopMatrix(); glutSwapBuffers();
}
void doFrame(int v)
  { frameNumber++;
  glutPostRedisplay();
  glutTimerFunc(30, doFrame, 0);
}
void init() {
  glClearColor(0.5f, 0.5f, 1, 1);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
```

```
  glOrtho(0, 7, -1, 4, -1, 1);
  glMatrixMode(GL_MODELVIEW);
} int main(int argc, char**
argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE);
  glutInitWindowSize(700, 500);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Rotating Windmill");
  init();
  glutDisplayFunc(display);
  glutTimerFunc(200, doFrame,
  0); glutMainLoop(); return 0;
}
```



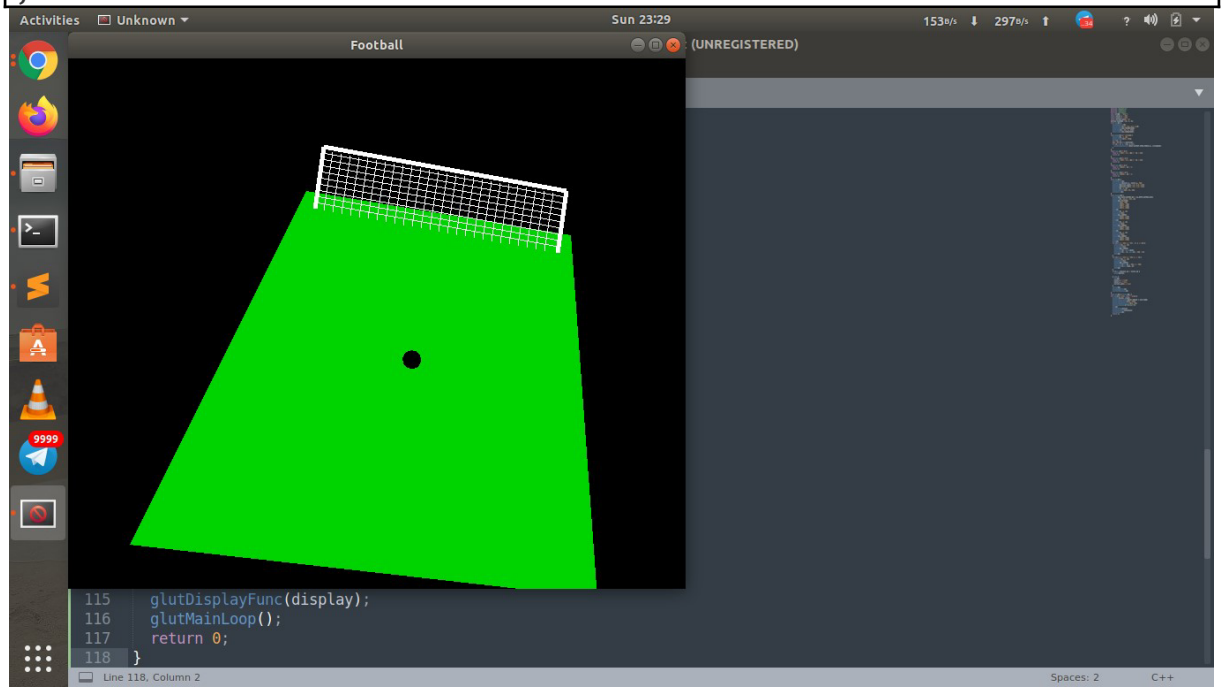
(iii) Simple animation of football goal

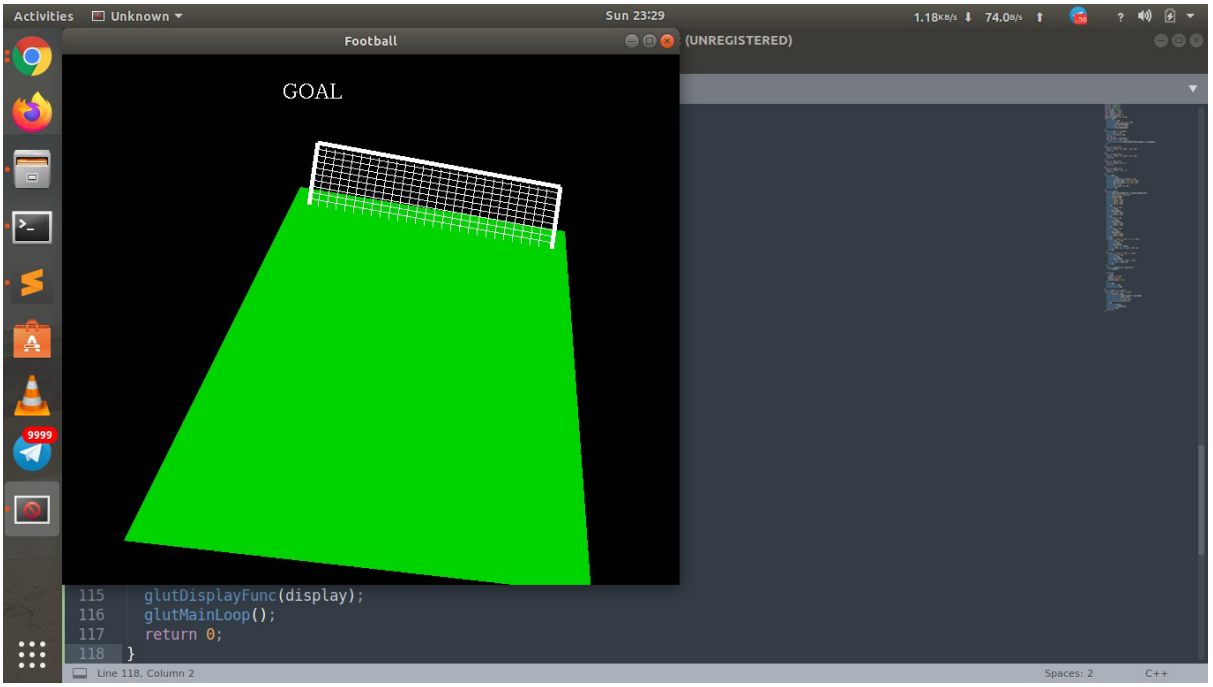
```
#include <GL/glut.h>
#include <stdlib.h>
#include <string.h> char
str1[] = "GOAL"; float
center_x = 300; float
center_y = 110; float
current_angle = 0; GLfloat
BLACK[] = {0, 0, 0};
void init() {
  glLoadIdentity();
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0, 700, 0, 600);
  glMatrixMode(GL_MODELVIEW);
} void output(char*
string) { glColor3f(1, 1,
1);
  glRasterPos2f(250,
  550); int len, i; len =
  (int)strlen(string); for
  (i = 0; i < len; i++) {
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, string[i]);
  }
} float cal_y(int
x) {
  float y = 6010 / 11 + ((-2 * x) / 11);
  return y;
} float cal_y1(int
x) {
  float y = 5290 / 11 + ((-2 * x) / 11);
  return y;
}
float cal_x(int y) {
  float x = (1530 + y) / 7;
  return x;
}
float cal_x1(int y) {
  float x = (3505 + y) / 7;
  return x;
}
void goal() {
  glPushMatrix();
  glTranslated(center_x, center_y,
  29); glRotatef(current_angle, 1.0,
  0.0, 0.0); glRotatef(current_angle,
  0.0, 1.0, 0.0); glColor3f(0, 0, 0);
  glutSolidSphere(30, 30, 30);
```

```
glPopMatrix();
```

```
} void
display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); glColor3f(0,
  212.0 / 255, 0);
  glBegin(GL_POLYGON);
  glVertex2f(70, 50);
  glVertex2f(600, -10);
  glVertex2f(570, 400);
  glVertex2f(270, 450); glEnd();
  glColor3f(1, 1, 1); glLineWidth(5);
  glBegin(GL_LINES);
  glVertex2f(280, 430);
  glVertex2f(290, 500); glEnd();
  glColor3f(1, 1, 1); glLineWidth(5);
  glBegin(GL_LINES);
  glVertex2f(555, 380);
  glVertex2f(565, 450); glEnd();
  glColor3f(1, 1, 1); glLineWidth(5);
  glBegin(GL_LINES);
  glVertex2f(290, 500);
  glVertex2f(565, 450); glEnd(); for
  (int i = 290; i < 555 - 5; i += 10) {
    glColor3f(1, 1, 1);
    glLineWidth(1);
    glBegin(GL_LINES);
    glVertex2f(i, cal_y1(i));
  glVertex2f(i + 10, cal_y(i + 10) +
  2); glEnd(); } for (int i = 443; i >
  380; i -= 7) {
    glColor3f(1, 1, 1);
    glLineWidth(1);
    glBegin(GL_LINES);
    glVertex2f(cal_x(i + 50), i + 50);
    glVertex2f(cal_x1(i), i);
    glEnd(); } if (cal_y1(center_x)
  < center_y) {
    output(str1);

  } else {
    goal();
```

```
    center_x += 0.06;
    center_y += 0.1;
    current_angle += 0.4;
  }
  glFlush();
  glutSwapBuffers();
  glutPostRedisplay();
}
void idle() { display(); } int
main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(700, 600);
  glutInitWindowPosition(10, 10);
  glutCreateWindow("Football");
  init();
  glutIdleFunc(idle);
  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```




Activities
Unknown
Sun 23:29
1.18KB/s
74.0B/s
Football
GOAL
(UNREGISTERED)
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
Line 118, Column 2
Spaces: 2
C++