//2.DDA Line Patterns

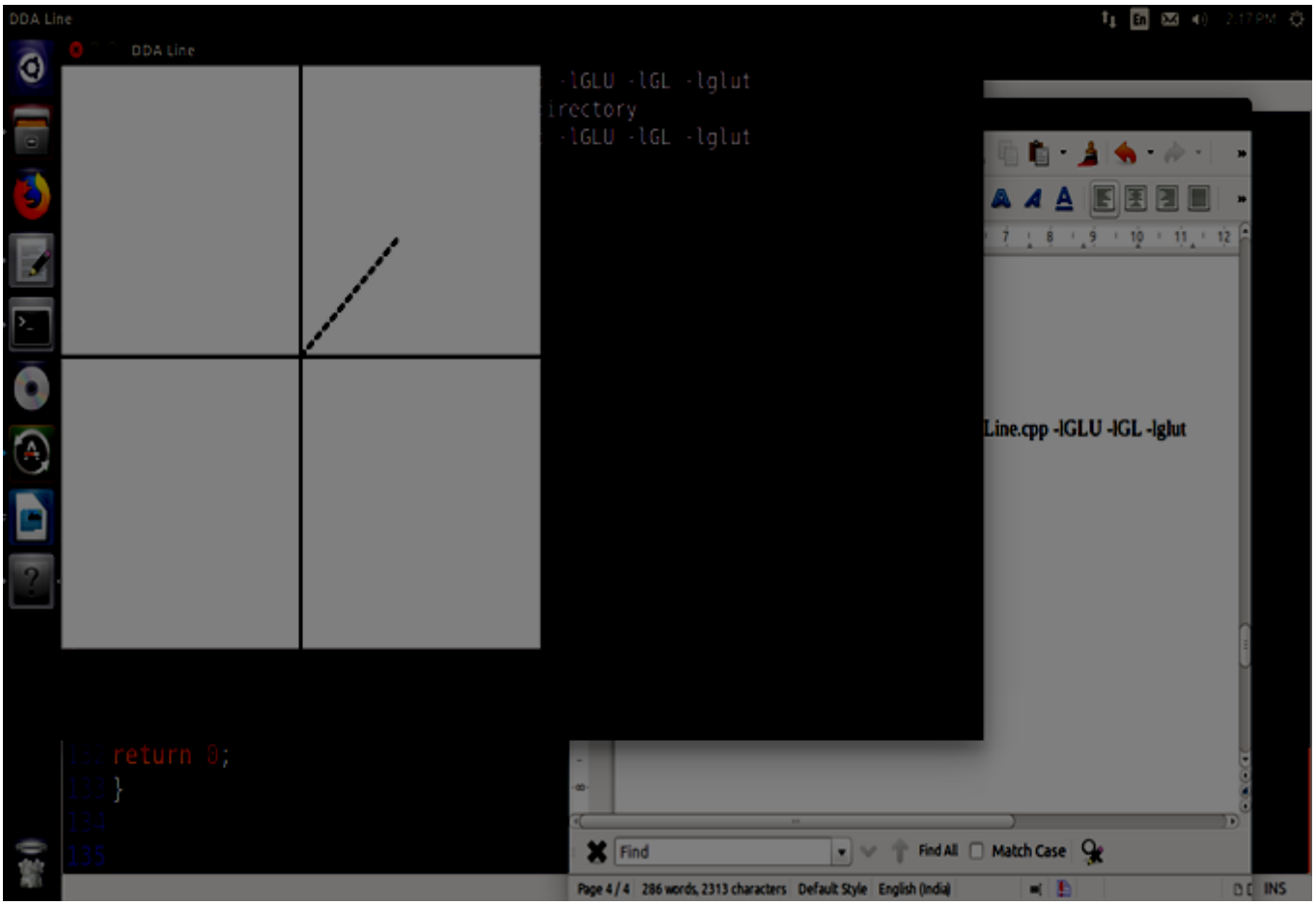
```
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
#include<iostream>
using namespace std;

float x1, x2, y1, y2;
int ch;

void init(){
        glClearColor(1,1,1,1);
        glColor3f(0,0,0);
        gluOrtho2D(-500,500,-500,500);
}

void display(){
        float dy, dx, step, x, y, Xin, Yin;

        glClear(GL_COLOR_BUFFER_BIT);
        glPointSize(3);
        glLineWidth(3);


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

        switch(ch){
                int i;
                case 1: {
                                for (i = 1; i <= step; i++) {
                                x = x + Xin;
                                y = y + Yin;
                                glBegin(GL_POINTS);
                                glVertex2i(x,  y);
                                glEnd();
                                }
```

```
                break;
            }
        case 2: {
                for (i = 1; i <= step; i++) {
                x = x + Xin;
                y = y + Yin;
                if(i%16<=8){
                        glBegin(GL_POINTS);
                        glVertex2i(x,  y);
                        glEnd();
                        }
                }
                break;
            }


        case 3: {
                for (i = 1; i <= step; i++) {
                x = x + Xin;
                y = y + Yin;
                if(i%8==0){
                        glBegin(GL_POINTS);
                        glVertex2i(x,  y);
                        glEnd();
                        }
                }break;
            }

        case 4: {
                for (i = 1; i <= step; i++) {
                x = x + Xin;
                y = y + Yin;

                if(i%16<=4){
                        glBegin(GL_POINTS);
                        glVertex2i(x,  y);
                        glEnd();
                        }
                if(i%8==2){
                        glBegin(GL_POINTS);
                        glVertex2i(x,  y);
                        glEnd();
                        }
                }break;
            }


}

glBegin(GL_LINES);
glVertex2i(-500,0);
glVertex2i(500,0);
```

```
        glVertex2i(0,-500);
        glVertex2i(0,500);
        glEnd();

        glFlush();
}

int main(int argc, char **argv){

        cout<<"Enter x1 and y1"<<endl;
        cin>>x1>>y1;
        cout<<"Enter x2 and y2"<<endl;
        cin>>x2>>y2;

        cout<<"1.Simple 2.Dashed 3.Dotted 4.Center"<<endl;
        cout<<"Enter your choice:"<<endl;
        cin>>ch;

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
        glutInitWindowPosition(0,0);
        glutInitWindowSize(500,500);

        glutCreateWindow("DDA Line");
        init();
        glutDisplayFunc(display);

        glutMainLoop();
return 0;
}
```

Output

```
aarti@aarti-Vostro-260s:~$  g++ DDALine.cpp  -lGLU  -lGL -lglut
aarti@aarti-Vostro-260s:~$  ./a.out
Enter  x1  and  y1
0  0
Enter  x2  and  y2
200  200
1.Simple  2.Dashed  3.Dotted  4.Center
Enter  your  choice:
2
```



//2.Bresenhams Line Patterns

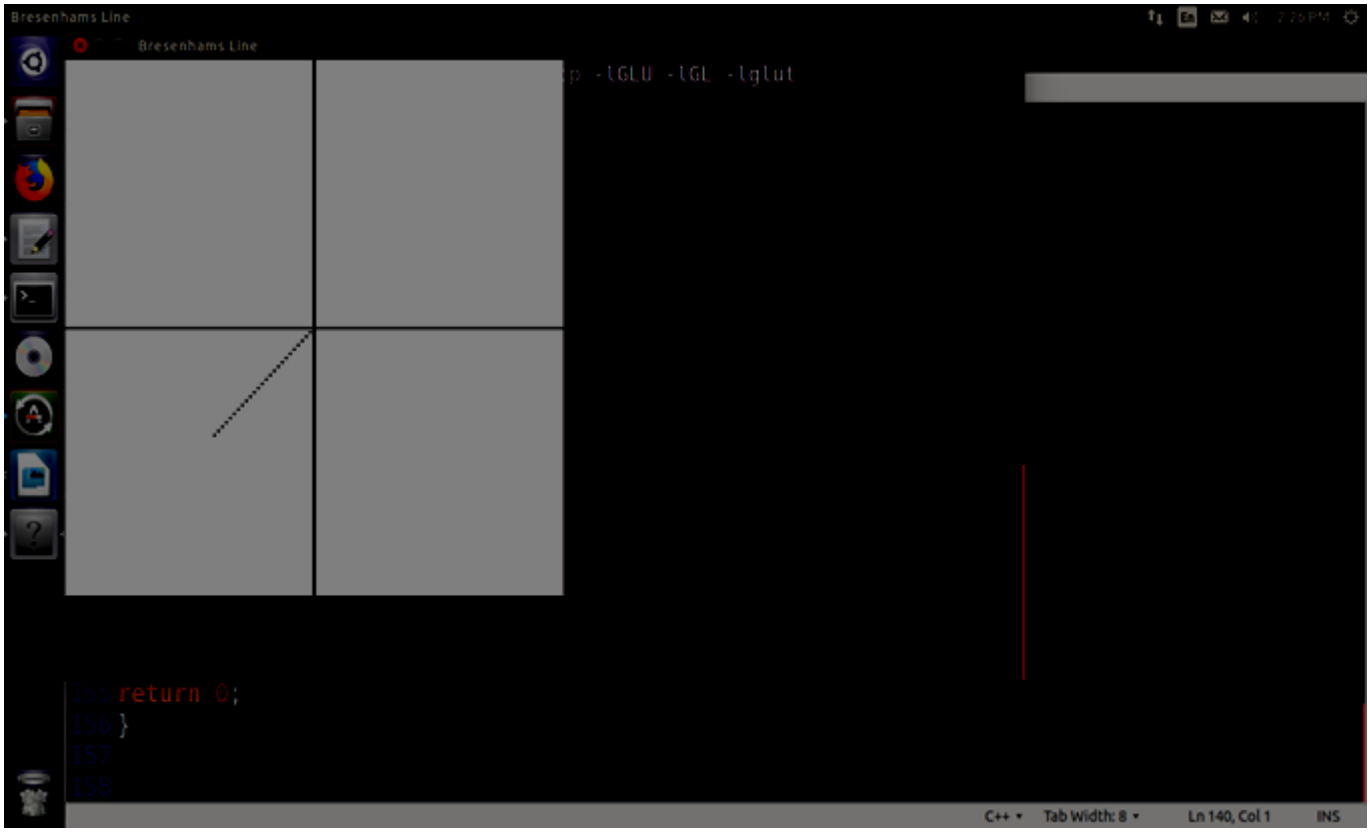
```
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
#include<iostream>
using namespace std;

int x1, x2, y1, y2;
int ch;

int sign(int a){
        if(a>0){return 1;}
        else if(a<0){return (-1);}
        else {return 0;}
}

void init(){
        glClearColor(1,1,1,1);
        glColor3f(0,0,0);
        gluOrtho2D(-500,500,-500,500);
}

void display(){
        int dy, dx, step, x, y, G, S1, S2;

        glClear(GL_COLOR_BUFFER_BIT);
        glPointSize(2);
        glLineWidth(2);

        dx = abs(x2 - x1);
        dy = abs(y2 - y1);

        if ((dx) > (dy)) {
          step = (dx);
        } else{
            step = (dy);
        }

        S1 = sign(x2-x1);
        S2 = sign(y2-y1);

        G = (2*dy)-dx;

          x = x1;
          y = y1;
          glBegin(GL_POINTS);
          glVertex2i(x,  y);
          glEnd();

        switch(ch){
                int i;
```

```
case 1: {
                for (i = 1; i <= step; i++) {
                while(G>=0){
                                y=y+S2;
                                G = G-(2*dx);
                        }
                x=x+S1;
                G=G+(2*dy);

                        glBegin(GL_POINTS);
                        glVertex2i(x,  y);
                        glEnd();
                }
                break;
        }
case 2: {
                for (i = 1; i <= step; i++) {
                while(G>=0){
                                y=y+S2;
                                G = G-(2*dx);
                        }
                x=x+S1;
                G=G+(2*dy);
                if(i%16<=8){
                        glBegin(GL_POINTS);
                        glVertex2i(x,  y);
                        glEnd();
                        }
                }
                break;
        }


case 3: {
                for (i = 1; i <= step; i++) {
                while(G>=0){
                                y=y+S2;
                                G = G-(2*dx);
                        }
                x=x+S1;
                G=G+(2*dy);
                if(i%8==0){
                        glBegin(GL_POINTS);
                        glVertex2i(x,  y);
                        glEnd();
                        }
                }break;
        }

case 4: {
                for (i = 1; i <= step; i++) {
                while(G>=0){
```

```
                                        y=y+S2;
                                        G = G-(2*dx);
                                }
                        x=x+S1;
                        G=G+(2*dy);
                        if(i%16<=4){
                                glBegin(GL_POINTS);
                                glVertex2i(x,  y);
                                glEnd();
                                }
                        if(i%8==2){
                                glBegin(GL_POINTS);
                                glVertex2i(x,  y);
                                glEnd();
                                }
                                        }break;
                }

        }

        glBegin(GL_LINES);
        glVertex2i(-500,0);
        glVertex2i(500,0);
        glVertex2i(0,-500);
        glVertex2i(0,500);
        glEnd();

        glFlush();
}

int main(int  argc, char **argv){

        cout<<"Enter  x1 and y1"<<endl;
        cin>>x1>>y1;
        cout<<"Enter  x2 and y2"<<endl;
        cin>>x2>>y2;

        cout<<"1.Simple  2.Dashed 3.Dotted 4.Center"<<endl;
        cout<<"Enter  your choice:"<<endl;
        cin>>ch;

        glutInit(&argc,  argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
        glutInitWindowPosition(0,0);
        glutInitWindowSize(500,500);
        glutCreateWindow("Bresenhams  Line");
        init();
        glutDisplayFunc(display);

        glutMainLoop();
        return  0;
}
```

Output

```
aarti@aarti-Vostro-260s:~$  g++ BreLine1.cpp  -lGLU -lGL -lglut
aarti@aarti-Vostro-260s:~$  ./a.out
Enter  x1 and y1
0 0
Enter  x2 and y2
-200 -200
1.Simple  2.Dashed 3.Dotted 4.Center
Enter  your choice:
3
```



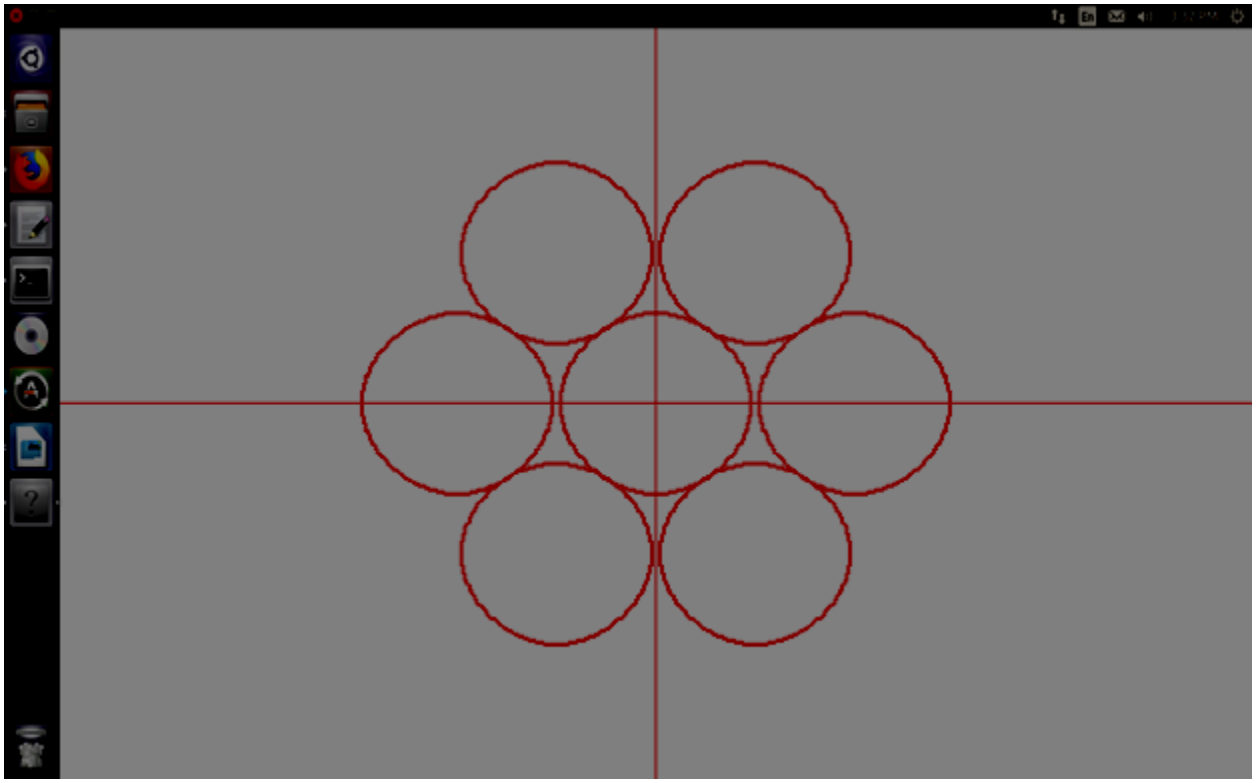
//Assignment 3-Bresenhams Circle

```
#include<iostream>
#include<GL/glut.h>
using namespace std;
int w = 600;
int h = 400;

void coordinate();
void myInit();
void MyDisplay();
void bresenhamCircle(int x,int y,int r);

int main(int a,char **v) {
glutInit(&a,v);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowPosition(0,0);
glutInitWindowSize(h,w);
glutCreateWindow("Bresenham Circle");
myInit();
glutDisplayFunc(MyDisplay);
glutMainLoop();
return 0;
}

void myInit() {
glPointSize(3.0);
glClearColor(1.0,1.0,1.0,0);
glColor3f(1.0,0.0,0.0);
gluOrtho2D(-w/2,w/2,-h/2,h/2);
}

void coordinate() {
glBegin(GL_LINES);
glVertex2d(-w/2,0);
glVertex2d(w/2,0);
glVertex2d(0,-h/2);
glVertex2d(0,h/2);
glEnd();
glFlush();
}

void MyDisplay() {
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
coordinate();
/* Cooncentric Circle
bresenhamCircle(0,0,50);
bresenhamCircle(0,0,100);
bresenhamCircle(0,0,150);*/

// Flower Pattern
```

```
bresenhamCircle(0,0,48);
bresenhamCircle(50,80,48);
bresenhamCircle(-50,80,48);
bresenhamCircle(-50,-80,48);
bresenhamCircle(50,-80,48);
bresenhamCircle(100,0,48);
bresenhamCircle(-100,0,48);
glFlush();
}

void bresenhamCircle(int x,int y,int r) {
int xi = 0;
int yi = r;
int pk = 3-2*r;
int pi = pk;
while(xi<=yi) {
glBegin(GL_POINTS);
glVertex2f(x+xi,y+yi);
glVertex2f(x+yi,y+xi);
glVertex2f(x+yi,y-xi);
glVertex2f(x+xi,y-yi);
glVertex2f(x-xi,y-yi);
glVertex2f(x-yi,y-xi);
glVertex2f(x-yi,y+xi);
glVertex2f(x-xi,y+yi);
glEnd();
glFlush();

if(pi<0) {
xi = xi+1;
yi = yi;
pi = pi+4*xi+6;
}
else {
xi = xi+1;
yi = yi-1;
pi = pi+4*(xi-yi)+10;
}
}
}
```

Output

```
aarti@aarti-Vostro-260s:~$  g++ BreCir.cpp  -lGLU  -lGL -lglut
aarti@aarti-Vostro-260s:~$  ./a.out
```



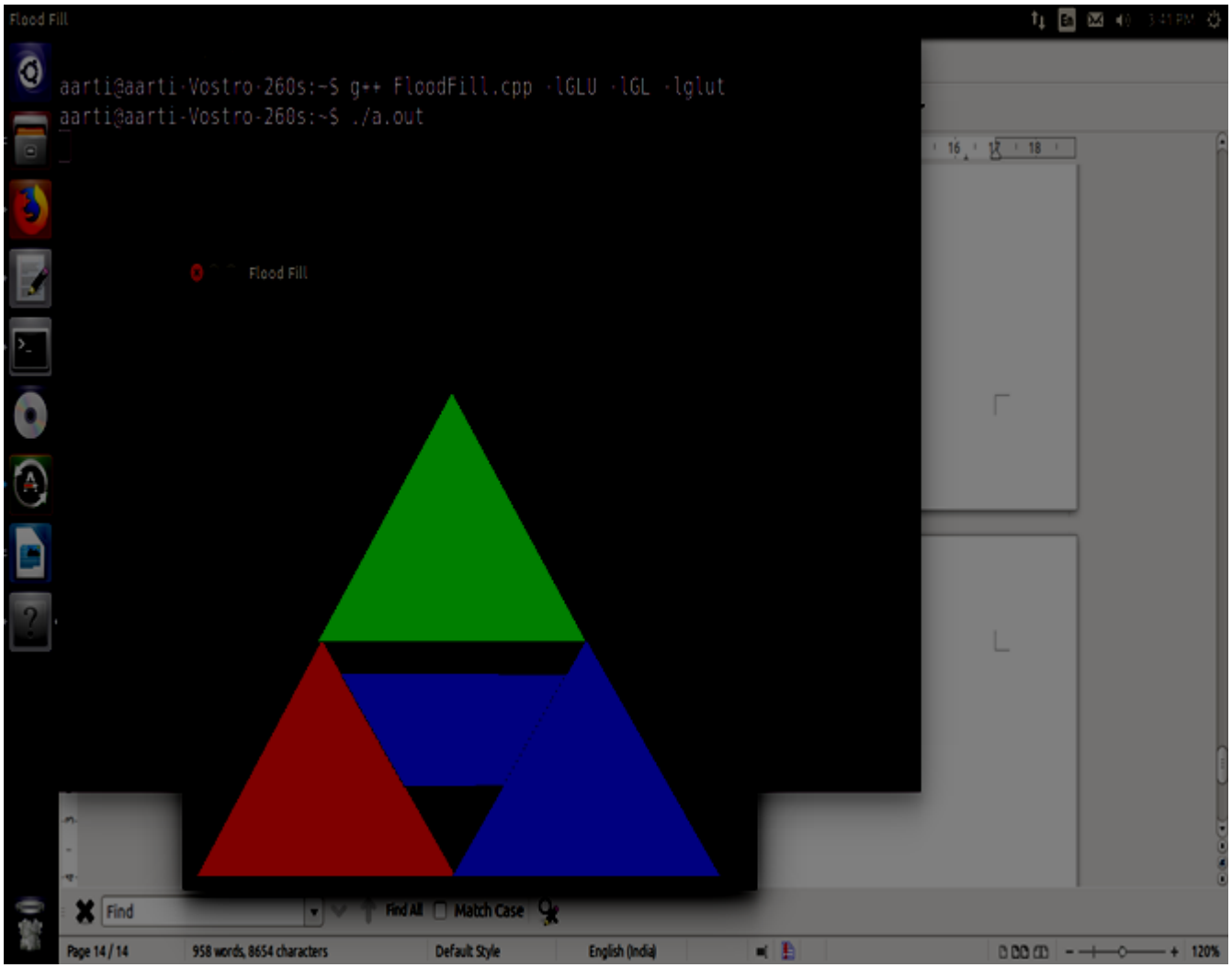
//4a.Flood Fill

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>

using namespace std;

void delay(float ms) {
   clock_t goal = ms + clock();
   while (goal > clock());
}

void init() {
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 640, 0, 480);
}

void flood_it(int x, int y, float *fillColor, float *ic) {
   float color[3];
   glReadPixels(x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, color);
   if (color[0] == ic[0] && color[1] == ic[1] && color[2] == ic[2]) {
      glColor3f(fillColor[0], fillColor[1], fillColor[2]);
      glBegin(GL_POINTS);
      glVertex2i(x, y);
      glEnd();
      glFlush();
      flood_it(x - 2, y, fillColor, ic);
      flood_it(x + 1, y, fillColor, ic);
      flood_it(x, y + 1, fillColor, ic);
      flood_it(x, y - 2, fillColor, ic);
   }
}

void mouse(int btn, int state, int x, int y) {
   y = 480 - y;
   if (btn == GLUT_LEFT_BUTTON) {
      if (state == GLUT_DOWN) {
         float intCol[] = {1, 0, 0};
         float color[] = {0, 0, 1};
         glReadPixels(x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, intCol);
         flood_it(x, y, color, intCol);
      }
   }
}

void world() {
   glPointSize(2);
   glClear(GL_COLOR_BUFFER_BIT);
```

```
    glColor3f(1, 0, 0);
    glBegin(GL_POLYGON);
    glVertex2i(15, 10);
    glVertex2i(155, 200);
    glVertex2i(305, 10);
    glEnd();
    glColor3f(0, 1, 0);
    glBegin(GL_POLYGON);
    glVertex2i(300, 398);
    glVertex2i(150, 198);
    glVertex2i(450, 198);
    glEnd();
    glColor3f(0, 0, 1);
    glBegin(GL_POLYGON);
    glVertex2i(300, 10);
    glVertex2i(600, 10);
    glVertex2i(450, 200);
    glEnd();
    glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Flood Fill");
    glutDisplayFunc(world);
    glutMouseFunc(mouse);
    init();
    glutMainLoop();
    return 0;
}
```

Output

```
aarti@aarti-Vostro-260s:~$  g++ FloodFill.cpp  -lGLU  -lGL -lglut
aarti@aarti-Vostro-260s:~$  ./a.out
```



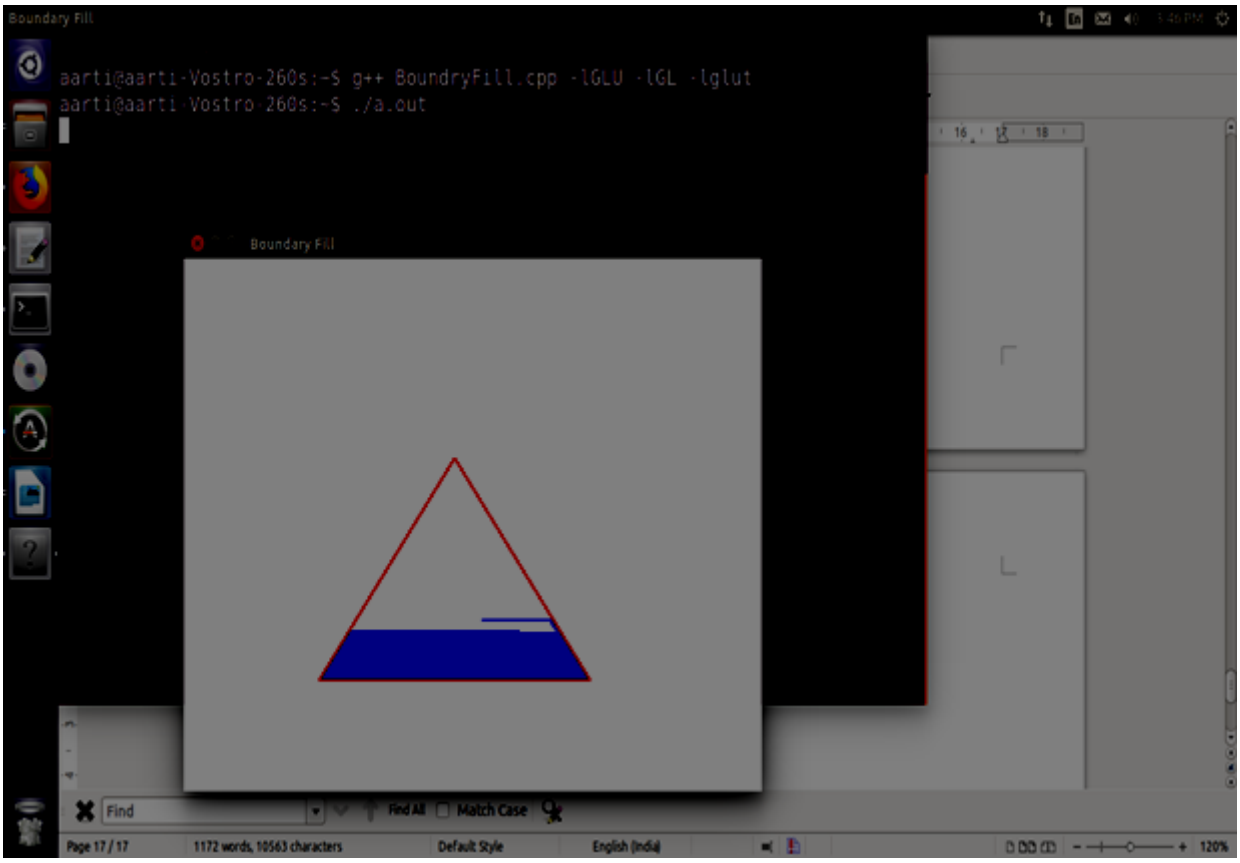
//4b.Boundary Fill

```
#include <iostream>
#include <math.h>
#include <time.h>
#include <GL/glut.h>

using namespace std;

void delay(float ms) {
   clock_t goal = ms + clock();
   while (goal > clock());
}

void init() {
   glClearColor(1.0, 1.0, 1.0, 0.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 640, 0, 480);
}

void bound_it(int x, int y, float *fillColor, float *bc) {
   float color[3];
   glReadPixels(x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, color);
   if ((color[0] != bc[0] || color[1] != bc[1] || color[2] != bc[2]) &&
     (color[0] != fillColor[0] || color[1] != fillColor[1] || color[2] != fillColor[2])) {
      glColor3f(fillColor[0], fillColor[1], fillColor[2]);
      glBegin(GL_POINTS);
      glVertex2i(x, y);
      glEnd();
      glFlush();
      bound_it(x + 1, y, fillColor, bc);
      bound_it(x - 2, y, fillColor, bc);
      bound_it(x, y + 2, fillColor, bc);
      bound_it(x, y - 2, fillColor, bc);
   }
}

void mouse(int btn, int state, int x, int y) {
   y = 480 - y;
   if (btn == GLUT_LEFT_BUTTON) {
      if (state == GLUT_DOWN) {
         float bCol[] = {1, 0, 0};
         float color[]= {0, 0, 1};
         bound_it(x, y, color, bCol);
      }
   }
}

void world() {
   glLineWidth(3);
   glPointSize(2);
```

```
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1, 0, 0);
    glBegin(GL_LINE_LOOP);
    glVertex2i(150, 100);
    glVertex2i(300, 300);
    glVertex2i(450, 100);
    glEnd();
    glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("Boundary Fill");
    glutDisplayFunc(world);
    glutMouseFunc(mouse);
    init();
    glutMainLoop();
    return 0;
}
```

Output

```
aarti@aarti-Vostro-260s:~$  g++ BoundryFill.cpp  -lGLU -lGL -lglut
aarti@aarti-Vostro-260s:~$  ./a.out
```



//5. Polygon Clipping

```
#include<stdio.h>
#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
#include<math.h>

typedef struct
{
        float x;
        float y;
}PT;
int n;

/*void drawpolygon(PT[],int);
void left(PT,PT[],PT[]);
void right(PT,PT[],PT[]);
void top(PT,PT[],PT[]);
void bottom(PT,PT[],PT[]);*/
int i,j;

PT d,p1,p2,p[20],pi1,pi2,pp[20];
void left()
{
i=0;j=0;

for(i=0;i<n;i++)
{
        if(p[i].x<p1.x  && p[i+1].x>=p1.x)  // Outside to Inside
        {
                if(p[i+1].x-p[i].x!=0)
                {
                        pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p1.x-p[i].x)+p[i].y;
                }
                else
                {
                pp[j].y=p[i].y;
                }
                pp[j].x=p1.x;
                j++;
                pp[j].x=p[i+1].x;
                pp[j].y=p[i+1].y;
                j++;
      }

if(p[i].x>=p1.x  && p[i+1].x>=p1.x)   //Inside  to Inside
{
        pp[j].y=p[i+1].y;
        pp[j].x=p[i+1].x;
        j++;
```

```
}
if(p[i].x>=p1.x  && p[i+1].x<p1.x)  //Inside  to Outside
{
        if(p[i+1].x-p[i].x!=0)
        {
                pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p1.x-p[i].x)+p[i].y;
        }
        else
        {
                pp[j].y=p[i].y;
        }
        pp[j].x=p1.x;
        j++;
}
}
for(i=0;i<j;i++)
{
        p[i].x=pp[i].x;
        p[i].y=pp[i].y;

}
p[i].x=pp[0].x;
p[i].y=pp[0].y;
n=j;
}

void  right()
{
i=0;j=0;

for(i=0;i<n;i++)
{
        if(p[i].x>p2.x  && p[i+1].x<=p2.x)  //Outside  to Inside
        {
                if(p[i+1].x-p[i].x!=0)
                {
                        pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p2.x-p[i].x)+p[i].y;
                }
        else
        {
                pp[j].y=p[i].y;
        }
        pp[j].x=p2.x;
        j++;
        pp[j].x=p[i+1].x;
        pp[j].y=p[i+1].y;
j++;
}
if(p[i].x<=p2.x&&p[i+1].x<=p2.x)  //Inside  to Inside
{

        pp[j].y=p[i+1].y;
```

```
		pp[j].x=p[i+1].x;
		j++;
}
if(p[i].x<=p2.x  && p[i+1].x>p2.x)  //Inside  to Outside
{
		if(p[i+1].x-p[i].x!=0)
		{
			pp[j].y=(p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)*(p2.x-p[i].x)+p[i].y;
		}
		else
		{
			pp[j].y=p[i].y;
		}
		pp[j].x=p2.x;
	j++;
}
}
for(i=0;i<j;i++)
{
p[i].x=pp[i].x;
p[i].y=pp[i].y;

}
p[i].x=pp[0].x;
p[i].y=pp[0].y;
n=j;
}

void top()
{
i=0;j=0;

for(i=0;i<n;i++)
{
		if(p[i].y>p2.y  && p[i+1].y<=p2.y)  //Outside  to Inside
		{
			if(p[i+1].y-p[i].y!=0)
			{
				pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p2.y-p[i].y)+p[i].x;
			}
			else
			{
				pp[j].x=p[i].x;
			}
			pp[j].y=p2.y;
			j++;

		pp[j].x=p[i+1].x;
		pp[j].y=p[i+1].y;
		j++;
		}
```

```
if(p[i].y<=p2.y  && p[i+1].y<=p2.y)  //Inside  to Inside
{
        pp[j].y=p[i+1].y;
        pp[j].x=p[i+1].x;
        j++;
}
if(p[i].y<=p2.y&&p[i+1].y>p2.y)  //Inside  to Outside
{

            if(p[i+1].y-p[i].y!=0)
            {
                pp[j].x=(p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)*(p2.y-p[i].y)+p[i].x;
        }
        else
        {
            pp[j].x=p[i].x;
          }
          pp[j].y=p2.y;
        j++;
}
}

for(i=0;i<j;i++)
{
p[i].x=pp[i].x;
p[i].y=pp[i].y;

}

p[i].x=pp[0].x;
p[i].y=pp[0].y;
n=j;
}


void bottom()
{
i=0;j=0;

for(i=0;i<n;i++)
{
        if(p[i].y<p1.y  && p[i+1].y>=p1.y)  //Outside  to Inside
        {
            if(p[i+1].y-p[i].y!=0)
            {
                pp[j].x=((p[i+1].x-p[i].x)/(p[i+1].y-p[i].y))*(p1.y-p[i].y)+p[i].x;
            }
        else
        {
            pp[j].x=p[i].x;
        }
        pp[j].y=p1.y;
```

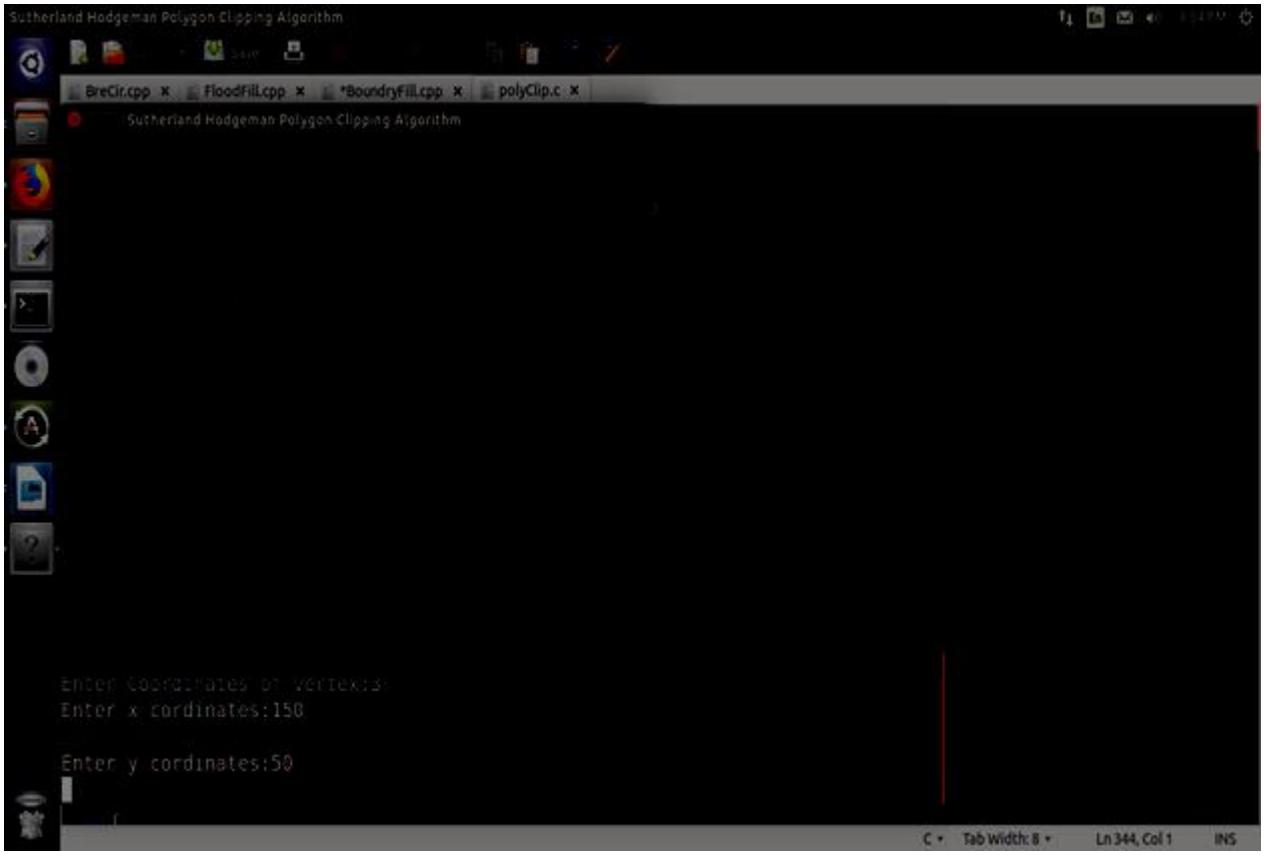
```
        j++;

        pp[j].x=p[i+1].x;
        pp[j].y=p[i+1].y;
        j++;
        }

if(p[i].y>=p1.y  && p[i+1].y>=p1.y)  //Inside  to Inside
{

        pp[j].x=p[i+1].x;
        pp[j].y=p[i+1].y;
j++;
}
if(p[i].y>=p1.y  && p[i+1].y<p1.y)  //Inside  to Outside
{
        if(p[i+1].y-p[i].y!=0)
        {
                pp[j].x=((p[i+1].x-p[i].x)/(p[i+1].y-p[i].y))*(p1.y-p[i].y)+p[i].x;
      }
        else
        {
        pp[j].x=p[i].x;

        }
pp[j].y=p1.y;
j++;
}
}
for(i=0;i<j;i++)
{
p[i].x=pp[i].x;
p[i].y=pp[i].y;

}

p[i].x=pp[0].x;
p[i].y=pp[0].y;
n=j;
}


void drawpolygon()
{
        glColor3f(1.0,0.0,0.0);
        for(i=0;i<n-1;i++)
        {
        glBegin(GL_LINES);
        glVertex2d(p[i].x,p[i].y);
        glVertex2d(p[i+1].x,p[i+1].y);
        glEnd();
        }
```

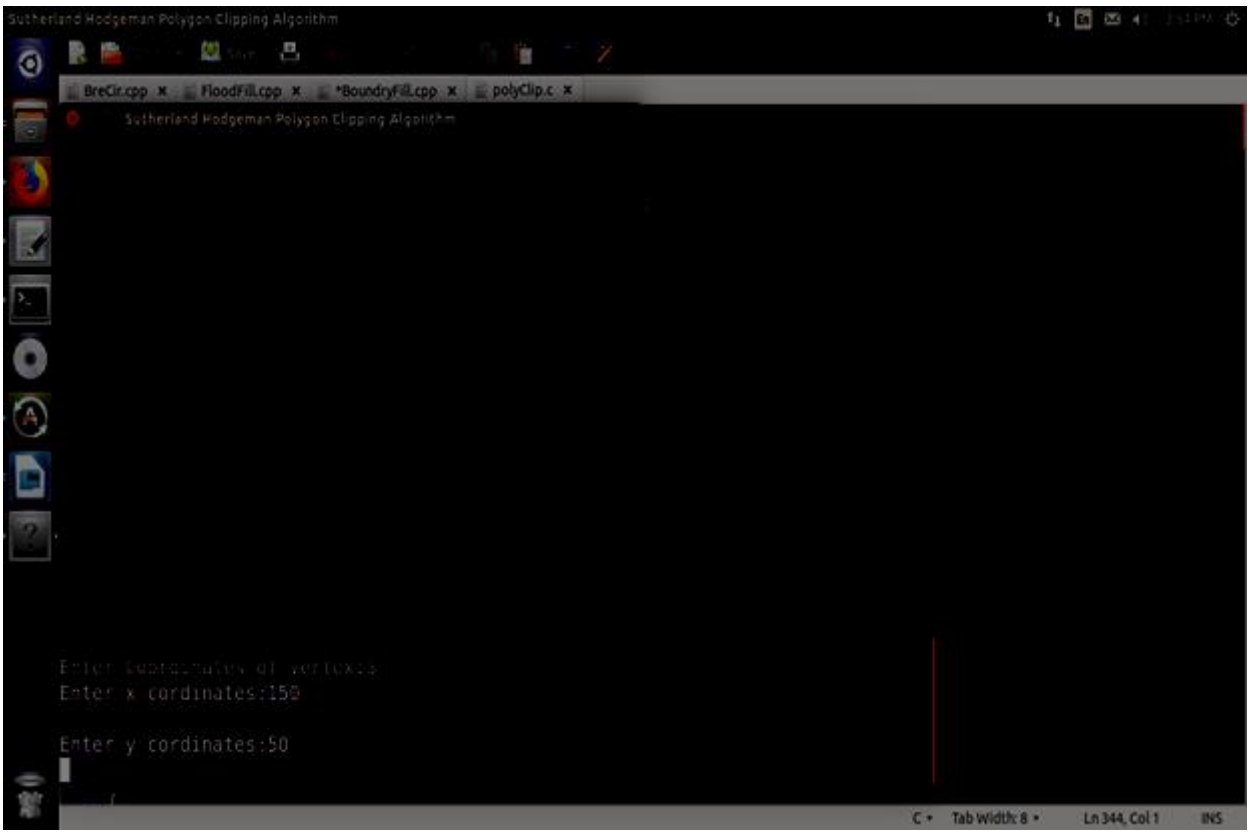
```
        glBegin(GL_LINES);
        glVertex2d(p[i].x,p[i].y);
        glVertex2d(p[0].x,p[0].y);
        glEnd();
}
void myMouse(int button,int state,int x,int y)
{
        if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
        {
                glClear(GL_COLOR_BUFFER_BIT);

                glBegin(GL_LINE_LOOP);
                glVertex2f(p1.x,p1.y);
                glVertex2f(p2.x,p1.y);
                glVertex2f(p2.x,p2.y);
                glVertex2f(p1.x,p2.y);
                glEnd();
                left();
                right();
                top();
                bottom();
                drawpolygon();
        }
        glFlush();
}
void display(void)
{
        //float x,y,dx,dy,length;
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.4,1.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(p1.x,p1.y);
        glVertex2f(p2.x,p1.y);
        glVertex2f(p2.x,p2.y);
        glVertex2f(p1.x,p2.y);
        glEnd();
        drawpolygon();
        glFlush();
}

void init(void)
{
        glClearColor(0.0,0.0,0.0,0.0);
        gluOrtho2D(0,500,0,500);
}
int main(int argc,char **argv)
{
        printf("Enter coordinates(left ,bottom)");

        printf("Enter x_coordinates:");
        scanf("%f",&p1.x);
        printf("Enter y_coordinates:");
```

```
        scanf("%f",&p1.y);
        printf("Enter  coordinates(right,top)");
        printf("Enter  x_coordinates:");
        scanf("%f",&p2.x);
        printf("Enter  y_coordinates:");
        scanf("%f",&p2.y);

        printf("\nEnter  the no of vertex:");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
                printf("\nEnter  Coordinates  of vertex:");
                printf("%d",i+1);
                printf("\nEnter  x cordinates:");
                scanf("%f",&p[i].x);
                printf("\nEnter  y cordinates:");
                scanf("%f",&p[i].y);
        }
        p[i].x=p[0].x;
        p[i].y=p[0].y;

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(640,480);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Sutherland  Hodgeman  Polygon  Clipping  Algorithm  ");
        init();
        //glClear(GL_COLOR_BUFFER_BIT);
        glutDisplayFunc(display);
        glutMouseFunc(myMouse);
        glFlush();
        glutMainLoop();
        return  0;

}
```

Output

```
aarti@aarti-Vostro-260s:~$ g++ polyClip.c -lGLU -lGL -lglut
aarti@aarti-Vostro-260s:~$ ./a.out
Enter coordinates(left ,bottom)Enter x_coordinates:100
Enter y_coordinates:100
Enter coordinates(right,top)Enter x_coordinates:300
Enter y_coordinates:300

Enter the no of vertex: 3

Enter Coordinates of vertex:1
Enter x cordinates:50

Enter y cordinates:50

Enter Coordinates of vertex:2
Enter x cordinates:150

Enter y cordinates:350

Enter Coordinates of vertex:3
Enter x cordinates:150

Enter y cordinates:50
```




Sutherland Hodgeman Polygon Clipping Algorithm
BreCir.cpp
FloodFill.cpp
*BoundryFill.cpp
polyClip.c
Sutherland Hodgeman Polygon Clipping Algorithm
Enter coordinates of vertex:1
Enter x cordinates:150
Enter y cordinates:50
C
Tab Width: 8
Ln 344, Col 1
INS

//6. 2D Transformations

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


#define MAX 10

        int n,op,tx,ty,angle,shx,shy,y,z,xr,yr;
        float a[MAX][3],b[3][3],c[MAX][3],sx,sy;


/* Function to plot a point */
void setPixel(GLint x, GLint y) {
  glBegin(GL_POINTS);
  glVertex2i(x,y);
  glEnd();
}

// Function of Rounding.
inline int round(const GLfloat a)
{
   return int(a+0.5);
}

// DDA Line Algorithm
void lineDDA(int x0,int y0,int xEnd,int yEnd)
{
   int dx = xEnd-x0;
   int dy = yEnd-y0;
   int steps,k;
   GLfloat xIncrement,yIncrement,x=x0,y=y0;

   if(abs(dx) > abs(dy))
      steps = abs(dx);
   else
      steps = abs(dy);

   xIncrement = GLfloat(dx) / GLfloat(steps);
   yIncrement = GLfloat(dy) / GLfloat(steps);
   setPixel(round(x),round(y));

   for(k=1;k<steps;k++)
   {
      x+= xIncrement;
      y+= yIncrement;
      setPixel(round(x),round(y));
   }
   glFlush();
}
```

```
void axis()
{
	int i,x,y;
	glClear(GL_COLOR_BUFFER_BIT);
	glClearColor(0.0,0.0,0.0,0);
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f (0.0, 1.0, 0.0);
	lineDDA(320,30,320,450);  //x axis
	lineDDA(20,240,620,240);  //y axis
	glColor3f (0.0, 1.0, 1.0);
	lineDDA(0,0,640,480);
	lineDDA(0,480,640,0);

	for(i=-10,x=20;i<=10;i++,x+=30) //horizontal line numbering
	{
		glColor3f (0.0, 1.0, 0.0);
		lineDDA(x,238,x,242);
	}

	for(i=7,y=30;i>=-7;i--,y+=30) //vertical line numbering
	{
		glColor3f (0.0, 1.0, 0.0);
		lineDDA(315,y,325,y);
	 }
}

void accept()
{
	int i;
	cout<<"Enter the coordinate : ";
	for(i=0;i<n;i++)
	{
		cout<<i+1<<","<<i+1<<":";
		cin>>a[i][0]>>a[i][1];
		a[i][2]=1;
	 }
	 a[n][0]=a[0][0];
	 a[n][1]=a[0][1];
	 a[n][2]=1;
}

void show()
{
	int i;
	axis();
	glColor3f (1.0, 0.0, 0.0);

      //  INPUT DATA
	for(i=0;i<n;i++)
```

```
lineDDA(320+a[i][0],240+a[i][1],320+a[(i+1)%n][0],240+a[(i+1)%n][1]);//homogeneous  form


        glColor3f  (1.0, 1.0, 1.0);

        //  OUTPUT DATA
        for(i=0;i<n;i++)
           lineDDA(320+c[i][0],240+c[i][1],320+c[(i+1)%n][0],240+c[(i+1)%n][1]);//homogeneous
form
}


void mul()
{
        int i,j,k;
        for(i=0;i<n;i++)
        {
                for(j=0;j<3;j++)
                {
                        c[i][j]=0;
                }
        }

        for(i=0;i<n;i++)
        {
                for(j=0;j<3;j++)
                {
                        for(k=0;k<3;k++)
                        {
                                c[i][j]=c[i][j]+a[i][k]*b[k][j];
                         }
                 }
         }



}


void scal()
{
        b[0][0]=sx;
        b[1][1]=sy;
        b[2][2]=1;
        b[0][1]=b[1][0]=b[1][2]=b[0][2]=b[2][1]=b[2][0]=0;
}


void rot_ref()
{
        float rad;
        int sign;
```

```
        rad=(angle*3.14)/180;

        if(z==1)
        sign=1;
        else
        sign=-1;
        b[0][0]=b[1][1]=cos(rad);
        b[2][2]=1;
        b[0][1]=sign*sin(rad);
        b[1][0]=-sign*sin(rad);
        b[2][0]=-xr*cos(rad)+yr*sign*sin(rad)+xr;
        b[2][1]=-xr*sin(rad)+yr*(-sign)*cos(rad)+yr;
        b[0][2]=b[1][2]=0;
}



void reflection()
{
        int i;
        b[2][2]=1;
        b[0][1]=b[1][0]=b[1][2]=b[0][2]=b[2][1]=b[2][0]=0;

        switch(z)
        {
                case 1 :  b[0][0]=1;
                          b[1][1]=-1;
                          break;

                case 2 :  b[0][0]=-1;
                          b[1][1]=1;
                          break;

                case 3 :  b[0][0]=-1;
                          b[1][1]=-1;
                          break;

                case 4 :  b[0][0]=0;
                          b[1][1]=0;
                          b[0][1]=1;
                          b[1][0]=1;
                          break;

                case 5 :  b[0][0]=0;
                          b[1][1]=0;
                          b[0][1]=-1;
                          b[1][0]=-1;
                          break;

        }
}
```

```
void choice(void)
{
                cout<<"\n\n**********   2D Transformations   *********";
                cout<<"\n\nEnter the no of vertices of polygon : ";
                cin>>n;
                accept();
                cout<<"\n**********MENU**********";
                cout<<"\n \n1) Scaling \n2) Rotation of arbitary \n3) Reflection";
                cout<<"\nEnter your choice : ";
                cin>>op;
                switch(op)
                {

                    case 1: cout<<"\nThe polygon before scaling";
                            cout<<"\nEnter the sx";
                            cin>>sx;
                            cout<<"\nEnter the sy";
                            cin>>sy;
                            scal();
                            mul();
                            show();
                            break;


                   case 2:  cout<<"\nThe polygon befor rotation";
                            cout<<"\nEnter the angle : ";
                            cin>>angle;
                            cout<<"\nPress 1 for anticlockwise and 2 for clockwise : ";
                            cin>>z;
                            cout<<"Enter the x and y coordinate : ";
                            cin>>xr>>yr;
                            rot_ref();
                            mul();
                            show();
                            break;



                       case 3: cout<<"\nThe polygon before Reflection";
                            cout<<"\n----------------------------";
                            cout<<"\n\t1. Against X-axis\n\t2. Against Y-axis\n\t3. Against
Origin";
                            cout<<"\n\t4. X = Y\n\t5. X = -Y\n\tEnter you Choice : ";
                            cin>>z;
                            reflection();
                            mul();
                            show();
                            break;
```

```
                default: cout<<"Invalid option";

            }
}



void init()
{
 /* Set the initial display mode */
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 /* Set the initial window position and size */
 glutInitWindowPosition(0,0);
 glutInitWindowSize(640,480);
 /* Create the window with title "2D Transformations" */
 glutCreateWindow("2D Transformations");
 /* Set clear color to white */
 glClearColor(1.0,1.0,1.0,1.0);
 /* Set fill color to black */
 glColor3f(0.0,0.0,0.0);
 /* glViewport(0 , 0 , 640 , 480); */
 /* glMatrixMode(GL_PROJECTION); */
 /* glLoadIdentity(); */
       glPointSize(1.0f);
 gluOrtho2D(0 , 640 , 0 , 480);


}

int main(int argc, char **argv)
{
 glutInit(&argc, argv);
 choice();

 init();
 glutDisplayFunc(show);

 glutMainLoop();
 return 0;
}
```

Output

```
aarti@aarti-Vostro-260s:~$  g++ Transformations.cpp  -lGLU  -lGL -lglut
aarti@aarti-Vostro-260s:~$  ./a.out


***********  2D  Transformations  *********

Enter  the no of vertices  of polygon  : 3
Enter  the coordinate  : 1,1:10
10
2,2:30 50
3,3:20 100

**********MENU**********

1) Scaling
2) Rotation  of arbitary
3) Reflection
Enter  your  choice  : 1

The  polygon  before  scaling
Enter  the sx 2

Enter  the sy 2
```

//7a.

Bezier Curve

```
#include  <iostream>
#include  <stdlib.h>
#include  <GL/glut.h>
#include  <math.h>
```

```
using namespace std;


//Point class for taking the points
class Point {
public:
   float x, y;
   void setxy(float x2, float y2)
   {
      x = x2; y = y2;
   }
   //operator overloading for '=' sign
   const Point & operator=(const Point &rPoint)
   {
      x = rPoint.x;
      y = rPoint.y;
      return *this;
   }

};

int factorial(int n)
{
   if (n<=1)
      return(1);
   else
      n=n*factorial(n-1);
   return n;
}

float binomial_coff(float n,float k)
{
   float ans;
   ans = factorial(n) / (factorial(k)*factorial(n-k));
   return ans;
}




Point abc[20];
int SCREEN_HEIGHT = 500;
int points = 0;
int clicks = 4;

void myInit() {
   glClearColor(1.0,1.0,1.0,0.0);
   glColor3f(0.0,0.0,0.0);
   glPointSize(3);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
```

```
    gluOrtho2D(0.0,640.0,0.0,500.0);

}

void drawDot(int  x, int y) {
    glBegin(GL_POINTS);
     glVertex2i(x,y);
    glEnd();
    glFlush();
}
void drawLine(Point  p1, Point p2) {
    glBegin(GL_LINES);
      glVertex2f(p1.x,  p1.y);
      glVertex2f(p2.x,  p2.y);
    glEnd();
    glFlush();
}

//Calculate  the bezier point
Point drawBezier(Point  PT[], double  t) {
Point P;
P.x=pow((1-t),3)*PT[0].x+3*t*pow((1-t),2)*PT[1].x+3*(1-t)*pow(t,2)*PT[2].x+pow(t,3)*PT[3].x;
P.y= pow((1-t),3)*PT[0].y+3*t*pow((1-t),2)*PT[1].y+3*(1-t)*pow(t,2)*PT[2].y+pow(t,3)*PT[3].y;

return P;
}

//Calculate  the bezier point [generalized]
Point drawBezierGeneralized(Point  PT[], double  t) {
       Point  P;
       P.x = 0; P.y = 0;
       for (int  i = 0; i<clicks;  i++)
       {
          P.x = P.x + binomial_coff((float)(clicks  - 1), (float)i)  * pow(t, (double)i)  * pow((1 - t),
(clicks  - 1 - i)) * PT[i].x;
          P.y = P.y + binomial_coff((float)(clicks  - 1), (float)i)  * pow(t, (double)i)  * pow((1 - t),
(clicks  - 1 - i)) * PT[i].y;
       }
       //cout<<P.x<<endl<<P.y;
       //cout<<endl<<endl;
       return  P;
    }


void myMouse(int  button,  int state, int x, int y) {
  // If left  button  was clicked
  if(button  == GLUT_LEFT_BUTTON  && state == GLUT_DOWN)  {
    // Store where mouse  was clicked,  Y is backwards.
    abc[points].setxy((float)x,(float)(SCREEN_HEIGHT  - y));
    points++;

    // Draw the red  dot.
```

```
    drawDot(x, SCREEN_HEIGHT - y);


    // If (click-amout) points are drawn do the curve.
    if(points == clicks)
    {
        glColor3f(0.2,1.0,0.0);
        // Drawing the control lines
        for(int k=0;k<clicks-1;k++)
            drawLine(abc[k], abc[k+1]);
       Point p1 = abc[0];
        /* Draw each segment of curve.Make t increment in smaller amounts for a more detailed
curve.*/
        for(double t = 0.0;t <= 1.0; t += 0.02)
        {
            Point p2 = drawBezierGeneralized(abc,t);
            cout<<p1.x<<" , "<<p1.y<<endl;
            cout<<p2.x<<" , "<<p2.y<<endl;
            cout<<endl;
            drawLine(p1, p2);
            p1 = p2;
        }
        glColor3f(0.0,0.0,0.0);

        points = 0;
    }
  }
}
void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,500);
    glutInitWindowPosition(100,150);
    glutCreateWindow("Bezier Curve");
    glutMouseFunc(myMouse);
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
   return 0;
}
```

Output

```
aarti@aarti-Vostro-260s:~$ g++ Bezier_new.cpp -lGLU -lGL -lglut
aarti@aarti-Vostro-260s:~$ ./a.out
65 , 163
```

65 , 163

65 , 163
74.6221 , 177.734

74.6221 , 177.734
84.286 , 191.588

84.286 , 191.588
93.9877 , 204.578

93.9877 , 204.578
103.723 , 216.721

103.723 , 216.721
113.489 , 228.036

113.489 , 228.036
123.281 , 238.538

.....

//7b. Koch Curve(Snowflake)

```
#include <GL/glut.h>
#include <math.h>

GLfloat oldx=-0.5,oldy=0.5;
```

```
void drawkoch(GLfloat dir,GLfloat len,int iter) {
        GLdouble dirRad = 0.0174533 * dir;
        GLfloat newX = oldx + len * cos(dirRad);
        float newY = oldy + len * sin(dirRad);
        if (iter==0) {
                glVertex2f(oldx, oldy);
                glVertex2f(newX, newY);
                oldx = newX;
                oldy = newY;
        }
        else {
                iter--;
                //draw the four parts of the side _/\_
                drawkoch(dir, len, iter);

                dir += 60.0;
                drawkoch(dir, len, iter);

                dir -= 120.0;
                drawkoch(dir, len, iter);

                dir += 60.0;
                drawkoch(dir, len, iter);
        }
}


void mydisplay()
{
         glClear( GL_COLOR_BUFFER_BIT );
         glBegin(GL_LINES);
        glColor3f(1.0, 0.0, 0.0);

        drawkoch(0.0,0.05,3);
        drawkoch(-120.0, 0.05, 3);
        drawkoch(120.0,0.05,3);

         glEnd();
         glFlush();
}


int main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Koch Snowflake");
        glutDisplayFunc(mydisplay);
        glutMainLoop();
}
```

Output

```
aarti@aarti-Vostro-260s:~$  g++ Koch_Curve.cpp  -lGLU  -lGL -lglut
aarti@aarti-Vostro-260s:~$  ./a.out
```

//8.Animation

```
#include<GL/glut.h>
#include<math.h>
```

```
#include<time.h>
#include<sys/timeb.h>
#define ESCAPE 27
int window;
float rtri = 0.0f;
//float rquad=0.0f;
void InitGL(int Width,int Height)
{
	glClearColor(0.0f,0.0f,0.0f,0.0f);//set window color
	//glClearDepth(1.0);
	/*glDepthFunc(GL_LESS);
	glEnable(GL_DEPTH_TEST);
	glShadeModel(GL_SMOOTH);*/
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(45.0f,(GLfloat)Width/(GLfloat)Height,0.1f,100.0f);
	glMatrixMode(GL_MODELVIEW);
}
float ballX=-0.5f;
float ballY=0.0f;
float ballZ=0.0f;
void drawBall(void)
{
	glColor3f(1.0,0.0,1.0);//set the ball color
	glTranslatef(ballX,ballY,ballZ);
	glRotatef(ballX,ballX,ballY,ballZ);
	glutSolidSphere(0.3,50,50);
	glTranslatef(ballX+1.5,ballY,ballZ);
	glutSolidSphere(0.3,50,50);
}
void DrawGLScene()
{
	//glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
	glClear(GL_COLOR_BUFFER_BIT);
	glLoadIdentity();
	glTranslatef(rtri,0.0f,-6.0f);
	glBegin(GL_POLYGON);
	glColor3f(1.0f,0.0f,0.0f);//set triangle color
	glVertex3f(-1.0f,1.0f,0.0);
	glVertex3f(0.4f,1.0f,0.0f);
	glVertex3f(1.0f,0.4f,0.0f);
	//glColor3f(0.0f,1.0f,0.0f);
	//glVertex3f(-1.0f,1.0f,0.0);
	//glColor3f(0.4f,0.0f,1.0f);
	//glVertex3f(1.0f,0.4f,0.0f);
	glEnd();

	drawBall();
	rtri+=0.005f;
	if(rtri>2)
	{
		rtri=-2.0f;
```

```
        }
        //rquad-=15.0f;
        glutSwapBuffers();
}
void keyPressed(unsigned char key,int x,int y)
{
        if(key==ESCAPE)
        {
                glutDestroyWindow(window);
                exit(0);
        }
}
int main(int argc,char **argv)
{
        glutInit(&argc,argv);
        //glutInitDisplayMode(GLUT_RGBA|GLUT_DOUBLE|GLUT_ALPHA|GLUT_DEPTH);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
        glutInitWindowSize(640,480);
        glutInitWindowPosition(0,0);
        window=glutCreateWindow("Moving Object");
        glutDisplayFunc(DrawGLScene);
        glutIdleFunc(DrawGLScene);
        glutKeyboardFunc(keyPressed);
        InitGL(640,480);
        glutMainLoop();
        return(0);
}
```

Output

```
aarti@aarti-Vostro-260s:~$ g++ Animation.cpp -lGLU -lGL -lglut
```

aarti@aarti-Vostro-260s:~$ ./a.out