

**ABESIT**

**COLLEGE CODE – 290**

# Lab File

|  |  |
| --- | --- |
| **NAME** | SANDEEP KUMAR SHUKLA |
| **BRANCH** | CSE |
| **UNIVERSITY ROLL NO.** | 1729010140 |
| **SESSION** | 2019-20 |
| **NAME OF LAB** | Computer Graphics Lab (RCS 653) |

**Aim :-** *Write a program to implement DDA algorithms for line and circle.*

**Code :-**

#include<graphics.h>

#include<conio.h>

#include<stdio.h>

#include<math.h>

//DDA Circle Logic

void DDA\_Circle()

{

int errorcode,tmp,i=1,rds;

float st\_x,st\_y,x1,x2,y1,y2,ep;

printf("Enter Radius:");

scanf("%d",&rds);

while(rds>pow(2,i))

i++;

ep=1/pow(2,i);

x1=rds; y1=0;

st\_x=rds; st\_y=0;

do

{ x2=x1+(y1\*ep);

y2=y1-(x2\*ep);

putpixel(x2+200,y2+200,10);

x1=x2;

y1=y2;

}while((y1-st\_y)<ep || (st\_x-x1)>ep);

}

// DDA Line Logic

void DDA\_Line()

{

float x, y,dx,dy,steps;

int x0, x1, y0, y1,i;

printf("Enter x0, y0 and x1, y1 of line :");

scanf("%d%d%d%d",&x0,&y0,&x1,&y1);

dx = (float)(x1 - x0);

dy = (float)(y1 - y0);

if(dx>=dy)

{

steps = dx;

}

else

{

steps = dy;

}

dx = dx/steps;

dy = dy/steps;

x = x0;

y = y0;

i = 1;

while(i<= steps)

{

putpixel(x, y, BLUE);

x += dx;

y += dy;

i=i+1;

}

}

//\*\*\*\*\*\*\*\*\*\*Main Fuction\*\*\*\*\*\*\*\*\*\*

int main()

{

int gd = DETECT ,gm, i;

initgraph(&gd, &gm, "");

DDA\_Line();

DDA\_Circle();

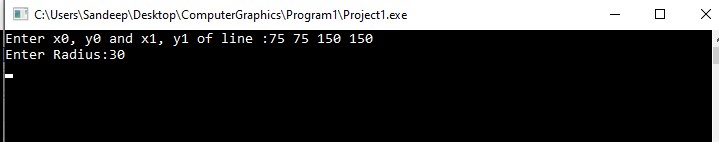
getch();

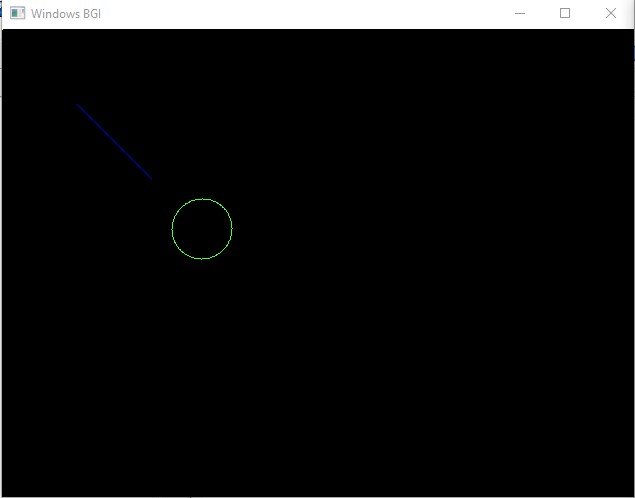
closegraph();

return 0;

}

**Output :-**

****

****

**Aim :-** *Write a program to implement Bresenham’s algorithms for line, circle and ellipse drawing.*

**Code :-**

#include<stdio.h>

#include<graphics.h>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bresenham's Line\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Draw line using Bresenham's Line Drawing Algorithm

void Bresenham\_Line()

{

int dx, dy, p, x, y ,error, x0, y0, x1, y1;

printf("Enter co-ordinates of first point: ");

scanf("%d%d", &x0, &y0);

printf("Enter co-ordinates of second point: ");

scanf("%d%d", &x1, &y1);

dx=x1-x0;

dy=y1-y0;

x=x0;

y=y0;

p=2\*dy-dx;

while(x<x1)

{

if(p>=0)

{

putpixel(x,y,7);

y=y+1;

p=p+2\*dy-2\*dx;

}

else

{

putpixel(x,y,7);

p=p+2\*dy;

}

x=x+1;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bresenham's Circle\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Draw Circle using Bresenham's Circle Drawing Algorithm

void drawCircle(int xc, int yc, int x, int y)

{

putpixel(xc+x, yc+y, RED);

putpixel(xc-x, yc+y, RED);

putpixel(xc+x, yc-y, RED);

putpixel(xc-x, yc-y, RED);

putpixel(xc+y, yc+x, RED);

putpixel(xc-y, yc+x, RED);

putpixel(xc+y, yc-x, RED);

putpixel(xc-y, yc-x, RED);

}

void Bresenham\_Circle()

{

int xc = 50, yc = 50, r = 30;

int x = 0, y = r;

int d = 3 - 2 \* r;

drawCircle(xc, yc, x, y);

while (y >= x)

{

// for each pixel we will

// draw all eight pixels

x++;

// check for decision parameter

// and correspondingly

// update d, x, y

if (d > 0)

{

y--;

d = d + 4 \* (x - y) + 10;

}

else

d = d + 4 \* x + 6;

drawCircle(xc, yc, x, y);

delay(50);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bresenham's Ellipse\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Draw Circle using Bresenham's Ellipse Drawing Algorithm

void Bresenham\_Ellipse()

{

long int d1,d2;

int i,x,y;

long int rx,ry,rxsq,rysq,tworxsq,tworysq,dx,dy;

printf("Enter the x Radius of the ellipse");

scanf("%ld",&rx);

printf("Enter the y Radius of the ellipse");

scanf("%ld",&ry);

rxsq=rx\*rx;

rysq=ry\*ry;

tworxsq=2\*rxsq;

tworysq=2\*rysq;

x=0;

y=ry;

d1=rysq - (rxsq \* ry) + (0.25 \* rxsq);

dx= tworysq \* x;

dy= tworxsq \* y;

do

{

putpixel(200+x,200+y,15);

putpixel(200-x,200-y,15);

putpixel(200+x,200-y,15);

putpixel(200-x,200+y,15);

if (d1 < 0)

{

x=x+1;

y=y;

dx=dx + tworysq;

d1=d1 + dx + rysq;

}

else

{

x=x+1;

y=y-1;

dx= dx + tworysq;

dy= dy - tworxsq;

d1= d1 + dx - dy + rysq;

}

delay(50);

}while (dx < dy);

d2 = rysq \* ( x + 0.5) \* ( x + 0.5 ) + rxsq \* (y - 1) \* (y-1) - rxsq \* rysq;

do

{

putpixel(200+x,200+y,15);

putpixel(200-x,200-y,15);

putpixel(200+x,200-y,15);

putpixel(200-x,200+y,15);

if (d2 >0)

{

x=x;

y=y-1;

dy = dy - tworxsq;

d2 = d2 - dy + rxsq;

}

else

{

x= x+1;

y=y-1;

dy=dy - tworxsq;

dx= dx + tworysq;

d2 = d2 + dx -dy + rxsq;

}

delay(50);

} while ( y> 0);

}

int main()

{

int gd=DETECT, gm;

initgraph(&gd, &gm, "");

Bresenham\_Line(); //for drawing Bresenham's Line

Bresenham\_Circle(); // for drawing Bresenham's Circle

Bresenham\_Ellipse(); // for drawing Bresenham's Ellipse

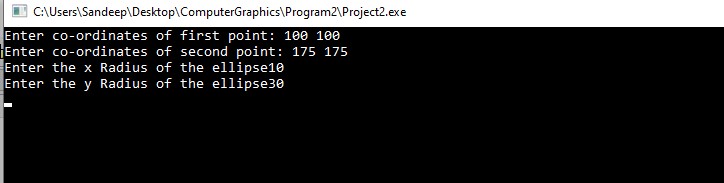
getch();

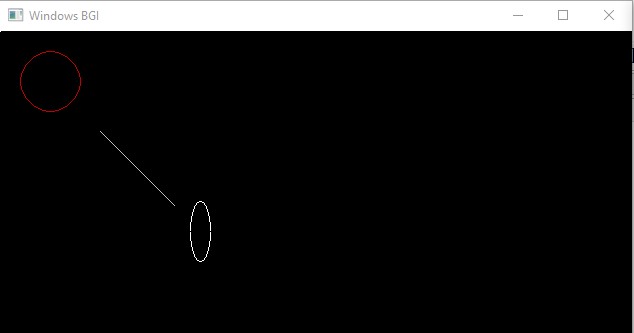
closegraph();

return 0;

}

**Output :-**

****

****

**Aim :-** *Write a program to implement Mid Point Circle algorithm using C .*

**Code :-**

#include<stdio.h>

#include<graphics.h>

void drawcircle(int x0, int y0, int radius)

{

int x = radius;

int y = 0;

int err = 0;

while (x >= y)

{

putpixel(x0 + x, y0 + y, 7);

putpixel(x0 + y, y0 + x, 7);

putpixel(x0 - y, y0 + x, 7);

putpixel(x0 - x, y0 + y, 7);

putpixel(x0 - x, y0 - y, 7);

putpixel(x0 - y, y0 - x, 7);

putpixel(x0 + y, y0 - x, 7);

putpixel(x0 + x, y0 - y, 7);

if (err <= 0)

{

y += 1;

err += 2\*y + 1;

}

if (err > 0)

{

x -= 1;

err -= 2\*x + 1;

}

}

}

int main()

{

int gd=DETECT, gm, error, x, y, r;

initgraph(&gd, &gm, "");

printf("Enter radius of circle: ");

scanf("%d", &r);

printf("Enter co-ordinates of center(x and y): ");

scanf("%d%d", &x, &y);

drawcircle(x, y, r);

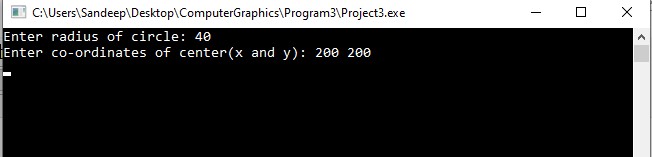
getch();

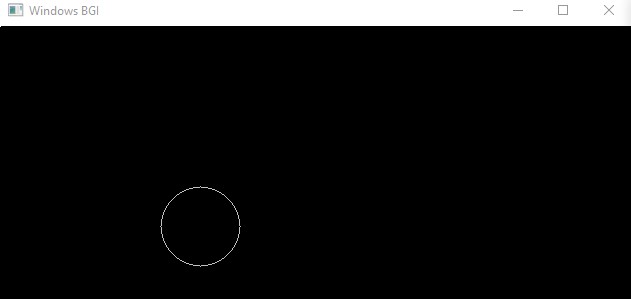
closegraph();

return 0;

}

**Output :-**

****

****

**Aim :-** *Write a program to implement Mid Point Ellipse algorithm using C .*

**Code :-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void ellipse(int xc,int yc,int rx,int ry)

{

int x, y, p;

x=0;

y=ry;

p=(ry\*ry)-(rx\*rx\*ry)+((rx\*rx)/4);

while((2\*x\*ry\*ry)<(2\*y\*rx\*rx))

{

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc-y,WHITE);

if(p<0)

{

x=x+1;

p=p+(2\*ry\*ry\*x)+(ry\*ry);

}

else

{

x=x+1;

y=y-1;

p=p+(2\*ry\*ry\*x+ry\*ry)-(2\*rx\*rx\*y);

}

}

p=((float)x+0.5)\*((float)x+0.5)\*ry\*ry+(y-1)\*(y-1)\*rx\*rx-rx\*rx\*ry\*ry;

while(y>=0)

{

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc-y,WHITE);

if(p>0)

{

y=y-1;

p=p-(2\*rx\*rx\*y)+(rx\*rx);

}

else

{

y=y-1;

x=x+1;

p=p+(2\*ry\*ry\*x)-(2\*rx\*rx\*y)-(rx\*rx);

}

}

}

int main()

{

int gm=DETECT,gd;

initgraph(&gm,&gd,"");

int xc,yc,rx,ry;

printf("Enter Xc=");

scanf("%d",&xc);

printf("Enter Yc=");

scanf("%d",&yc);

printf("Enter Rx=");

scanf("%d",&rx);

printf("Enter Ry=");

scanf("%d",&ry);

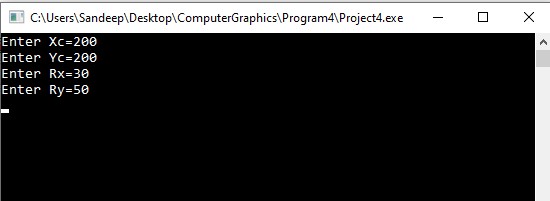
ellipse(xc,yc,rx,ry);

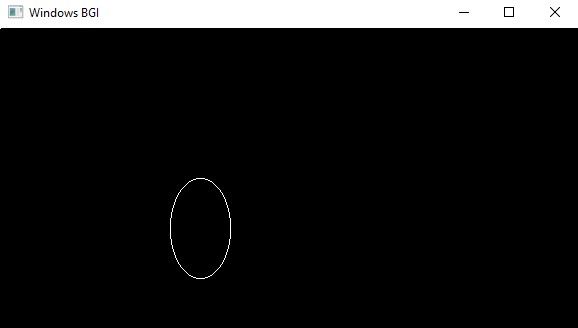
getch();

closegraph();

}

**Output :-**

****

****

**Aim :-** *Write a program to perform 2D Transformations such as translation, rotation, scaling, reflection and sharing.*

**Code :-**

#include<graphics.h>

#include<stdio.h>

#include<conio.h>

#include<process.h>

#include<math.h>

void Translation();

void translateLine ( int P[][2], int T[]);

void Rotation();

void TriAngle(int x1, int y1, int x2, int y2, int x3, int y3);

void Rotate(int x1, int y1, int x2, int y2, int x3, int y3);

void Scaling();

void scale(int x[], int y[], int sx, int sy);

void Reflection();

void DrawFn();

void FlipV();

void FlipH();

int main()

{

int T;

do{

printf("Please select your task to perform: \n");

printf(" 1. Translation \n 2. Rotation \n 3. Scaling \n 4. Reflection \n 5. Exit \n");

scanf("%d",&T);

switch(T)

{

case 1 :

Translation();

break;

case 2 :

Rotation();

break;

case 3 :

Scaling();

break;

case 4 :

Reflection();

break;

case 5 :

exit(1);

default :

printf("Invalid Choice!!");

}

}while(T!=5);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Translation Function Start\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void Translation()

{

int P[2][2] = {100, 100,200 , 200}; // coordinates of point

int T[] = {10, 50}; // translation factor

translateLine (P, T);

}

// function to translate line

void translateLine ( int P[][2], int T[])

{

/\* init graph and line() are used for

representing line through graphical

functions

\*/

int gd = DETECT, gm, errorcode;

initgraph (&gd, &gm, "");

// drawing original line using graphics functions

setcolor (2);

line(P[0][0], P[0][1], P[1][0], P[1][1]);

// calculating translated coordinates

P[0][0] = P[0][0] + T[0];

P[0][1] = P[0][1] + T[1];

P[1][0] = P[1][0] + T[0];

P[1][1] = P[1][1] + T[1];

// drawing translated line using graphics functions

setcolor(3);

line(P[0][0], P[0][1], P[1][0], P[1][1]);

getch();

cleardevice();

getch();

closegraph();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Translation Function End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Rotation Function Start\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void Rotation()

{

int gd = DETECT, gm;

int x1, y1, x2, y2, x3, y3;

initgraph(&gd, &gm, " ");

printf("Enter the 1st point for the triangle:");

scanf("%d%d", &x1, &y1);

printf("Enter the 2nd point for the triangle:");

scanf("%d%d", &x2, &y2);

printf("Enter the 3rd point for the triangle:");

scanf("%d%d", &x3, &y3);

TriAngle(x1, y1, x2, y2, x3, y3);

getch();

//cleardevice();

Rotate(x1, y1, x2, y2, x3, y3);

//setcolor(1);

TriAngle(x1, y1, x2, y2, x3, y3);

getch();

}

void TriAngle(int x1, int y1, int x2, int y2, int x3, int y3) {

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

}

void Rotate(int x1, int y1, int x2, int y2, int x3, int y3) {

int x, y, a1, b1, a2, b2, a3, b3, p = x2, q = y2;

float Angle;

printf("Enter the angle for rotation:");

scanf("%f", &Angle);

//cleardevice();

Angle = (Angle \* 3.14) / 180;

a1 = p + (x1 - p) \* cos(Angle)-(y1 - q) \* sin(Angle);

b1 = q + (x1 - p) \* sin(Angle)+(y1 - q) \* cos(Angle);

a2 = p + (x2 - p) \* cos(Angle)-(y2 - q) \* sin(Angle);

b2 = q + (x2 - p) \* sin(Angle)+(y2 - q) \* cos(Angle);

a3 = p + (x3 - p) \* cos(Angle)-(y3 - q) \* sin(Angle);

b3 = q + (x3 - p) \* sin(Angle)+(y3 - q) \* cos(Angle);

printf("Rotate");

TriAngle(a1, b1, a2, b2, a3, b3);

getch();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Rotation Function End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Scaling Function Start\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void Scaling()

{

int x[] = { 100, 200, 300 };

int y[] = { 200, 100, 200 };

int sx = 2, sy = 2;

int gd, gm;

initgraph(&gd, &gm," ");

scale(x, y, sx,sy);

getch();

}

void findNewCoordinate(int s[][2], int p[][1])

{

int temp[2][1] = { 0 };

for (int i = 0; i < 2; i++)

for (int j = 0; j < 1; j++)

for (int k = 0; k < 2; k++)

temp[i][j] += (s[i][k] \* p[k][j]);

p[0][0] = temp[0][0];

p[1][0] = temp[1][0];

}

void scale(int x[], int y[], int sx, int sy)

{

// Triangle before Scaling

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

// Initializing the Scaling Matrix.

int s[2][2] = { sx, 0, 0, sy };

int p[2][1];

// Scaling the triangle

for (int i = 0; i < 3; i++)

{

p[0][0] = x[i];

p[1][0] = y[i];

findNewCoordinate(s, p);

x[i] = p[0][0];

y[i] = p[1][0];

}

// Triangle after Scaling

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Scaling Function End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Reflection Function Start\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int graDriver=DETECT,graMode;

int n,xs[100],ys[100],i;

int tempYaxis,tempXaxis;

void Reflection()

{

printf("Enter number of sides: ");

scanf("%d",&n);

printf("Enter co-rdinates: x,y for each point ");

for(i=0;i<n;i++)

scanf("%d%d",&xs[i],&ys[i]);

initgraph(&graDriver,&graMode,"");

setcolor(RED);

DrawFn();//original

FlipV();

setcolor(BLUE);

DrawFn();//vertical flip

FlipH();

setcolor(GREEN);

DrawFn();//Horizontal flip

getch();

}

void DrawFn()

{

for(i=0;i<n;i++)

line(xs[i],ys[i],xs[(i+1)%n],ys[(i+1)%n]);

}

void FlipV()

{

tempXaxis=getmaxy()/2;

for(i=0;i<n;i++)

ys[i]=tempXaxis+(tempXaxis-ys[i]);

//drawing horizontal axis to flip about

for(i=0;i<getmaxx();i++)

putpixel(i,tempXaxis,WHITE);

}

void FlipH()

{

tempYaxis=getmaxx()/2;

for(i=0;i<n;i++)

xs[i]=tempYaxis+(tempYaxis-xs[i]);

setcolor(WHITE);

//drawing vertical axis

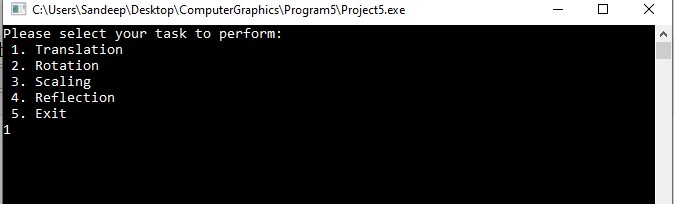
for(i=0;i<getmaxy();i++)

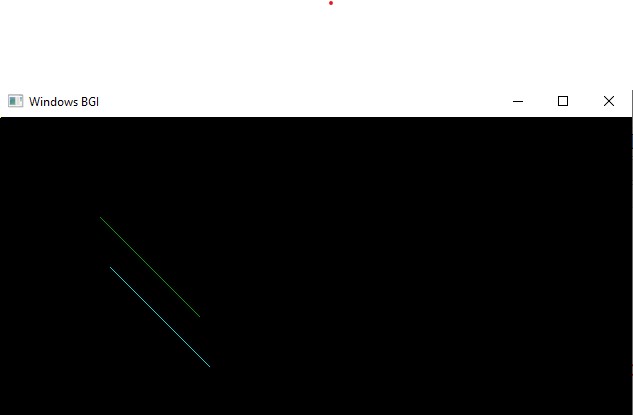
putpixel(tempYaxis,i,WHITE);

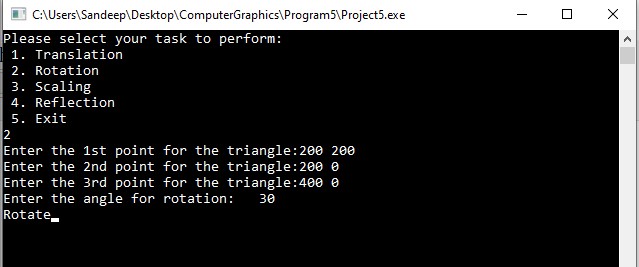
}

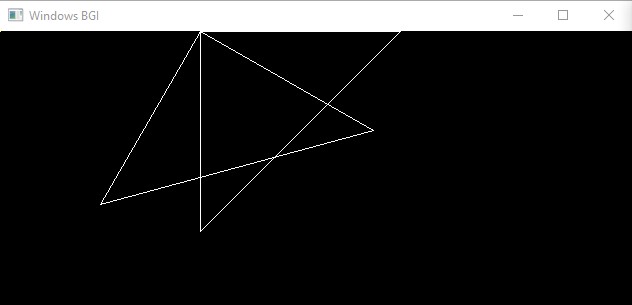
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Reflection Function End\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

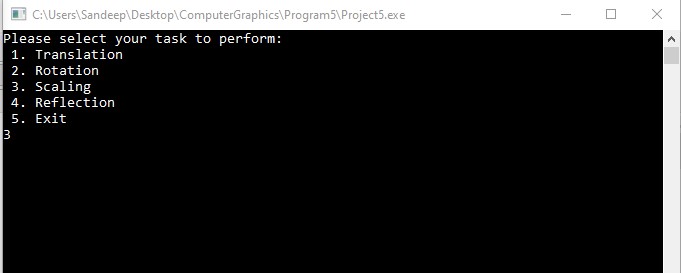
**Output :-**

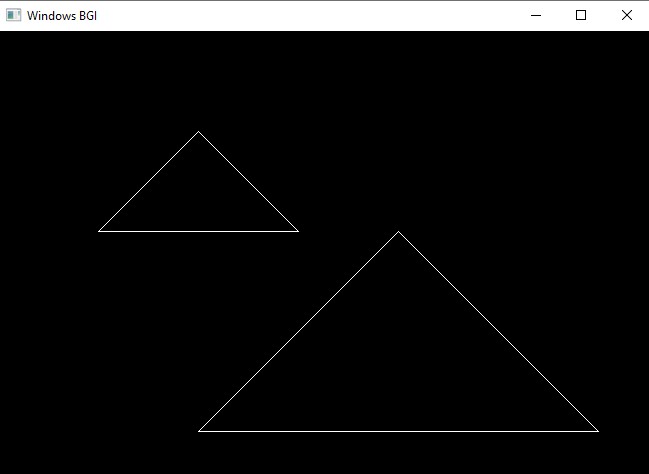
****

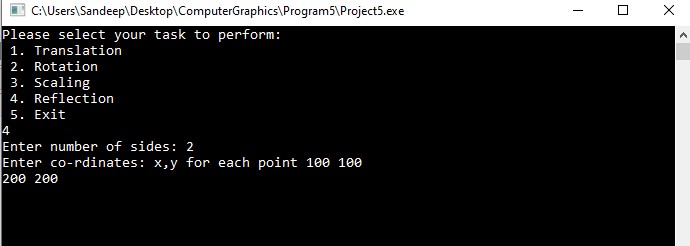
****

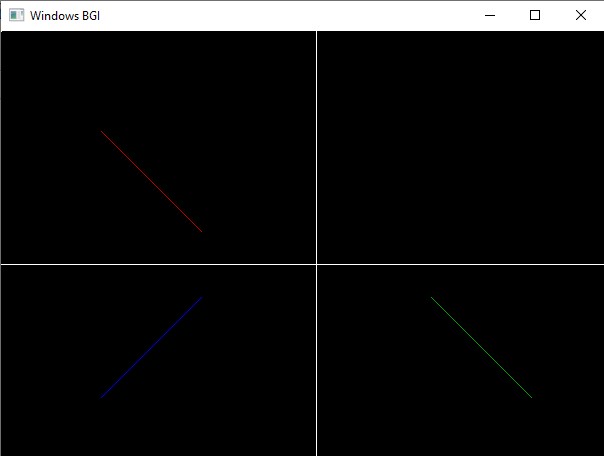
****

****

****

****

****

****

**Aim :-** *Write a program to implement Cohen–Sutherland 2D clipping and window–viewport mapping.*

**Code :-**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<graphics.h>

#include<dos.h>

typedef struct coordinate

{

int x,y;

char code[4];

}PT;

void drawwindow();

void drawline(PT p1,PT p2);

PT setcode(PT p);

int visibility(PT p1,PT p2);

PT resetendpt(PT p1,PT p2);

int main()

{

int gd=DETECT,v,gm;

PT p1,p2,p3,p4,ptemp;

printf("\nEnter x1 and y1\n");

scanf("%d %d",&p1.x,&p1.y);

printf("\nEnter x2 and y2\n");

scanf("%d %d",&p2.x,&p2.y);

initgraph(&gd,&gm,"c:\\turboc3\\bgi");

drawwindow();

delay(500);

drawline(p1,p2);

delay(500);

cleardevice();

delay(500);

p1=setcode(p1);

p2=setcode(p2);

v=visibility(p1,p2);

delay(500);

switch(v)

{

case 0: drawwindow();

delay(500);

drawline(p1,p2);

break;

case 1: drawwindow();

delay(500);

break;

case 2: p3=resetendpt(p1,p2);

p4=resetendpt(p2,p1);

drawwindow();

delay(500);

drawline(p3,p4);

break;

}

delay(5000);

closegraph();

}

void drawwindow()

{

line(150,100,450,100);

line(450,100,450,350);

line(450,350,150,350);

line(150,350,150,100);

}

void drawline(PT p1,PT p2)

{

line(p1.x,p1.y,p2.x,p2.y);

}

PT setcode(PT p) //for setting the 4 bit code

{

PT ptemp;

if(p.y<100)

ptemp.code[0]='1'; //Top

else

ptemp.code[0]='0';

if(p.y>350)

ptemp.code[1]='1'; //Bottom

else

ptemp.code[1]='0';

if(p.x>450)

ptemp.code[2]='1'; //Right

else

ptemp.code[2]='0';

if(p.x<150)

ptemp.code[3]='1'; //Left

else

ptemp.code[3]='0';

ptemp.x=p.x;

ptemp.y=p.y;

return(ptemp);

}

int visibility(PT p1,PT p2)

{

int i,flag=0;

for(i=0;i<4;i++)

{

if((p1.code[i]!='0') || (p2.code[i]!='0'))

flag=1;

}

if(flag==0)

return(0);

for(i=0;i<4;i++)

{

if((p1.code[i]==p2.code[i]) && (p1.code[i]=='1'))

flag='0';

}

if(flag==0)

return(1);

return(2);

}

PT resetendpt(PT p1,PT p2)

{

PT temp;

int x,y,i;

float m,k;

if(p1.code[3]=='1')

x=150;

if(p1.code[2]=='1')

x=450;

if((p1.code[3]=='1') || (p1.code[2]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(p1.y+(m\*(x-p1.x)));

temp.y=k;

temp.x=x;

for(i=0;i<4;i++)

temp.code[i]=p1.code[i];

if(temp.y<=350 && temp.y>=100)

return (temp);

}

if(p1.code[0]=='1')

y=100;

if(p1.code[1]=='1')

y=350;

if((p1.code[0]=='1') || (p1.code[1]=='1'))

{

m=(float)(p2.y-p1.y)/(p2.x-p1.x);

k=(float)p1.x+(float)(y-p1.y)/m;

temp.x=k;

temp.y=y;

for(i=0;i<4;i++)

temp.code[i]=p1.code[i];

return(temp);

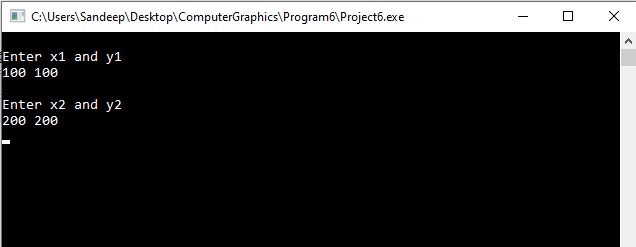
}

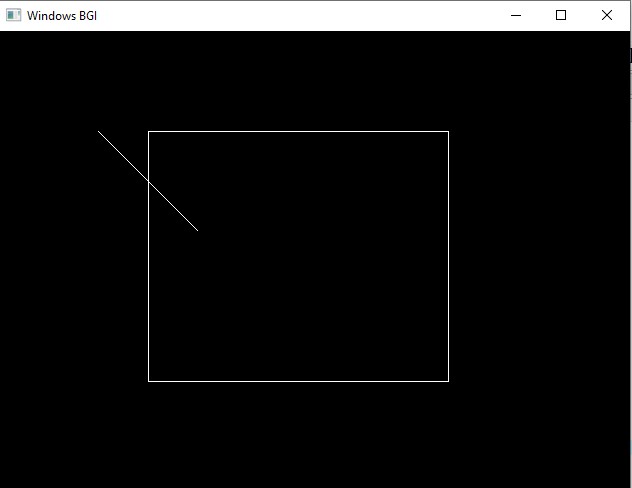
else

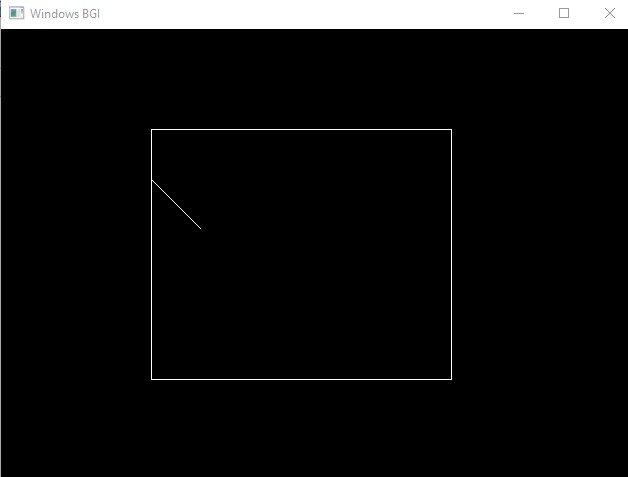
return(p1);

}

**Output :-**

****

****

****

**Aim :-** *Write a program to implement Liang Barksy Line Clipping Algorithm.*

**Code :-**

#include<stdio.h>

#include<graphics.h>

#include<math.h>

#include<dos.h>

int main()

{

int i,gd=DETECT,gm;

int x1,y1,x2,y2,xmin,xmax,ymin,ymax,xx1,xx2,yy1,yy2,dx,dy;

float t1,t2,p[4],q[4],temp;

x1=120;

y1=120;

x2=300;

y2=300;

xmin=100;

ymin=100;

xmax=250;

ymax=250;

initgraph(&gd,&gm,"");

rectangle(xmin,ymin,xmax,ymax);

dx=x2-x1;

dy=y2-y1;

p[0]=-dx;

p[1]=dx;

p[2]=-dy;

p[3]=dy;

q[0]=x1-xmin;

q[1]=xmax-x1;

q[2]=y1-ymin;

q[3]=ymax-y1;

for(i=0;i<4;i++)

{

if(p[i]==0)

{

printf("line is parallel to one of the clipping boundary");

if(q[i]>=0)

{

if(i<2)

{

if(y1<ymin)

{

y1=ymin;

}

if(y2>ymax)

{

y2=ymax;

}

line(x1,y1,x2,y2);

}

if(i>1)

{

if(x1<xmin)

{

x1=xmin;

}

if(x2>xmax)

{

x2=xmax;

}

line(x1,y1,x2,y2);

}

}

}

}

t1=0;

t2=1;

for(i=0;i<4;i++)

{

temp=q[i]/p[i];

if(p[i]<0)

{

if(t1<=temp)

t1=temp;

}

else

{

if(t2>temp)

t2=temp;

}

}

if(t1<t2)

{

xx1 = x1 + t1 \* p[1];

xx2 = x1 + t2 \* p[1];

yy1 = y1 + t1 \* p[3];

yy2 = y1 + t2 \* p[3];

line(xx1,yy1,xx2,yy2);

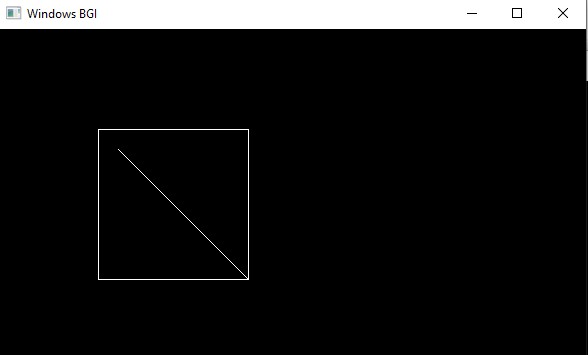
}

getch();

closegraph();

}

**Output :-**

****

**Aim :-** *To perform 3D Transformations such as translation, rotation and scaling.*

**Code :-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

int maxx,maxy,midx,midy;

void axis()

{

getch();

cleardevice();

line(midx,0,midx,maxy);

line(0,midy,maxx,midy);

}

int main()

{

int gd,gm,x,y,z,ang,x1,x2,y1,y2;

detectgraph(&gd,&gm);

initgraph(&gd,&gm,"C:/TC/BGI");

setfillstyle(3,25);

maxx=getmaxx();

maxy=getmaxy();

midx=maxx/2;

midy=maxy/2;

outtextxy(100,100,"ORIGINAL OBJECT");

line(midx,0,midx,maxy);

line(0,midy,maxx,midy);

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

axis();

outtextxy(100,20,"TRANSLATION");

printf("\n\n Enter the Translation vector: ");

scanf("%d%d",&x,&y);

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

bar3d(midx+(x+100),midy-(y+20),midx+(x+60),midy-(y+90),20,5);

axis();

outtextxy(100,20,"SCALING");

printf("\n Enter the Scaling Factor: ");

scanf("%d%d%d",&x,&y,&z);

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

bar3d(midx+(x\*100),midy-(y\*20),midx+(x\*60),midy-(y\*90),20\*z,5);

axis();

outtextxy(100,20,"ROTATION");

printf("\n Enter the Rotation angle: ");

scanf("%d",&ang);

x1=100\*cos(ang\*3.14/180)-20\*sin(ang\*3.14/180);

y1=100\*sin(ang\*3.14/180)+20\*sin(ang\*3.14/180);

x2=60\*cos(ang\*3.14/180)-90\*sin(ang\*3.14/180);

y2=60\*sin(ang\*3.14/180)+90\*sin(ang\*3.14/180);

axis();

printf("\n After rotating about z-axis\n");

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

bar3d(midx+x1,midy-y1,midx+x2,midy-y2,20,5);

axis();

printf("\n After rotating about x-axis\n");

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

bar3d(midx+100,midy-x1,midx+60,midy-x2,20,5);

axis();

printf("\n After rotating about y-axis\n");

bar3d(midx+100,midy-20,midx+60,midy-90,20,5);

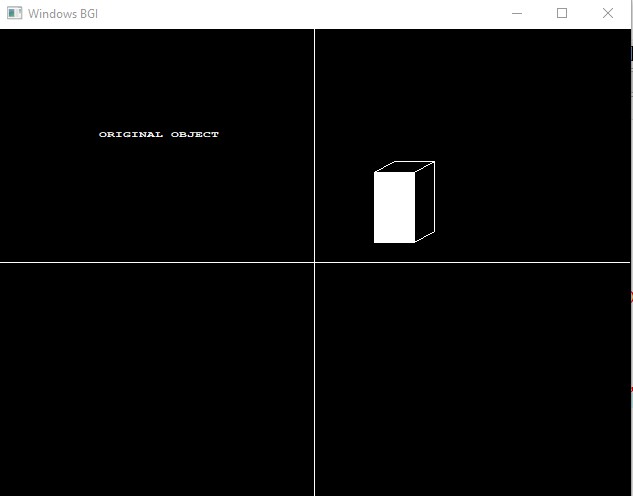
bar3d(midx+x1,midy-20,midx+x2,midy-90,20,5);

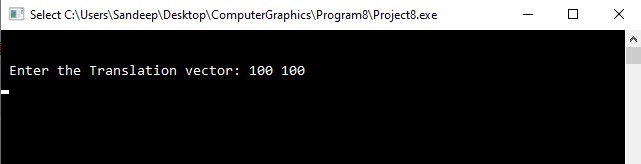
axis();

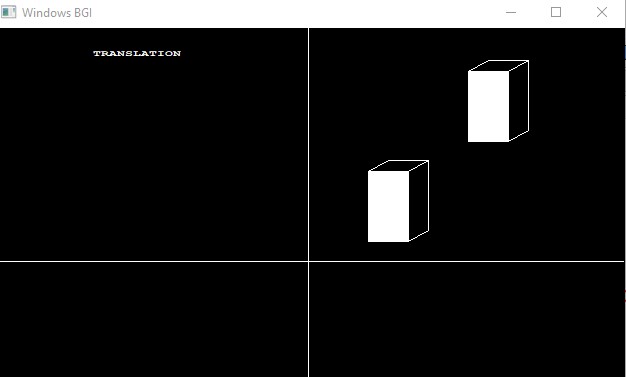
closegraph();

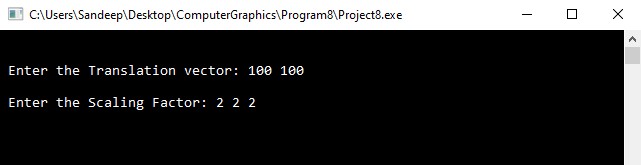
}

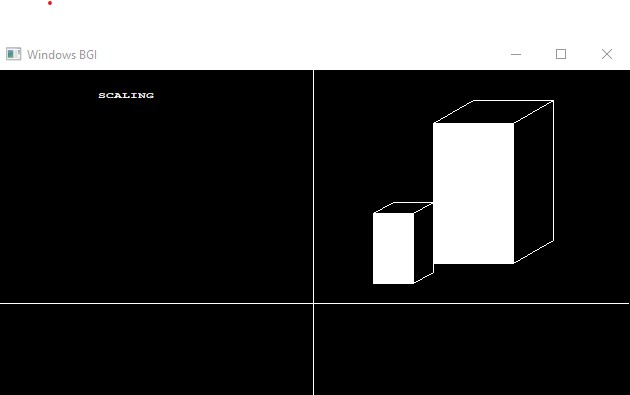
**Output :-**

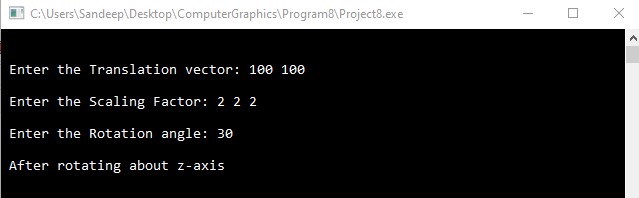
****

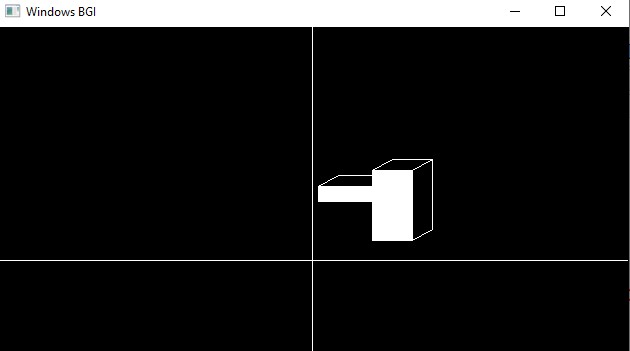
****

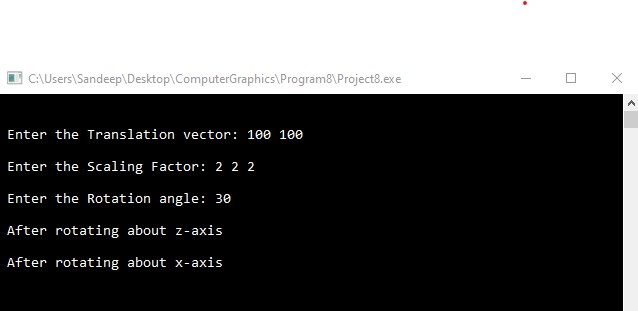
****

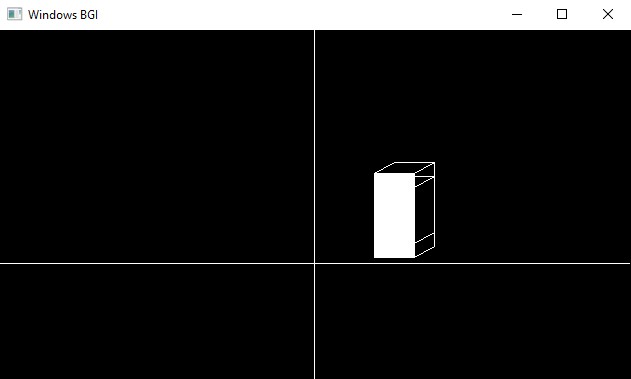
****

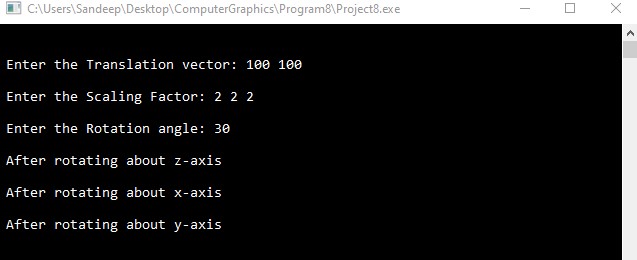
****

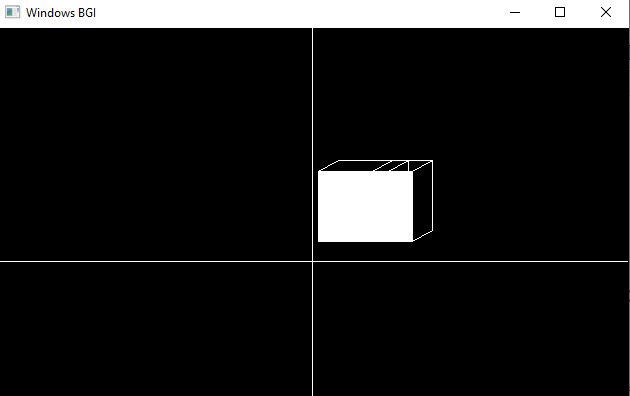
****

****

****

****

****

****

**Aim :-** *Write a program to convert between color models.*

**Code :-**

#include<stdio.h>

#include<conio.h>

#define MIN(a,b) (a<b?a:b)

#define MAX(a,b) (a>b?a:b)

#define NO\_HUE -1

void rgbtohsv(float r,float g,float b)

{

float h,s,v;

float max=MAX(r,MAX(g,b)),min=MIN(r,MIN(g,b));

float delta=max-min;

v=max;

if(max!=0.0)

s=delta/max;

else

s=0.0;

if(s==0.0)

h=NO\_HUE;

else

{

if(r==max)

h=(g-b)/delta;

else if(g==max)

h=2+(b-r)/delta;

else if(b==max)

h=4+(r-g)/delta;

h\*=60.0;

if(h<0)

h+=360.0;

h/=360.0;

}

printf("\n H=%f\n S=%f\n V=%f",h,s,v);

}

int main()

{

float a,b,c;

printf("\n Enter the RGB values:\n");

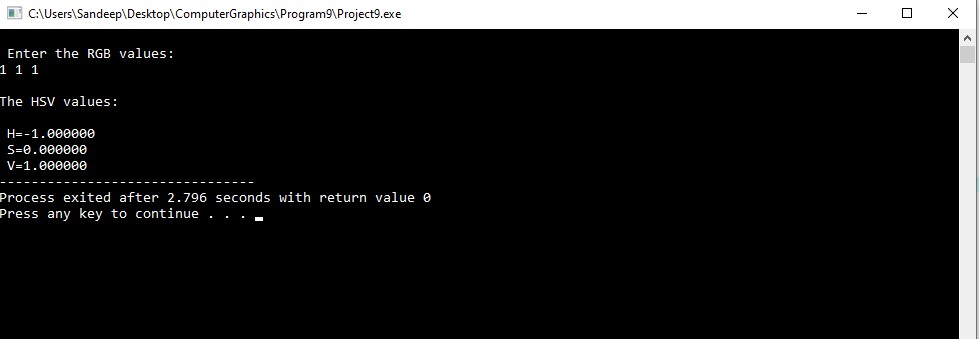
scanf("%f%f%f",&a,&b,&c);

printf("\nThe HSV values:\n");

rgbtohsv(a,b,c);

}

**Output :-**

****