

CG-Practical File

**Sub- Computer Graphics**

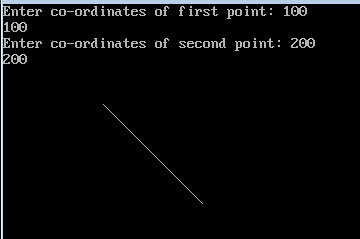
**Name- SUMIT KUMAR**

**Course- B.Sc. Hons. CS Roll No – 19/78055**

**Sem – VI Year- III**

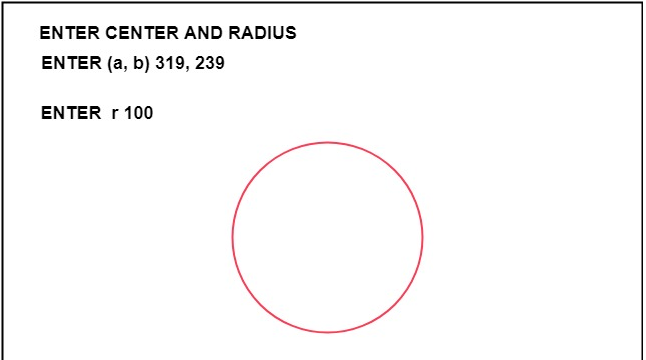
# Q1: Write a program to implement Bresenhams line drawing algorithm.

1. #include<stdio.h>
2. #include<graphics.h>
3. **void** drawline(**int** x0, **int** y0, **int** x1, **int** y1)
4. {
5. **int** dx, dy, p, x, y;
6. dx=x1-x0;
7. dy=y1-y0;
8. x=x0;
9. y=y0;
10. p=2\*dy-dx;
11. **while**(x<x1)
12. {
13. **if**(p>=0)
14. {
15. putpixel(x,y,7);
16. y=y+1;
17. p=p+2\*dy-2\*dx;
18. }
19. **else**
20. {
21. putpixel(x,y,7);
22. p=p+2\*dy;}
23. x=x+1;
24. }
25. }
26. **int** main()
27. {
28. **int** gdriver=DETECT, gmode, error, x0, y0, x1, y1;
29. initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
30. printf("Enter co-ordinates of first point: ");
31. scanf("%d%d", &x0, &y0);
32. printf("Enter co-ordinates of second point: ");
33. scanf("%d%d", &x1, &y1);
34. drawline(x0, y0, x1, y1);
35. **return** 0;
36. }



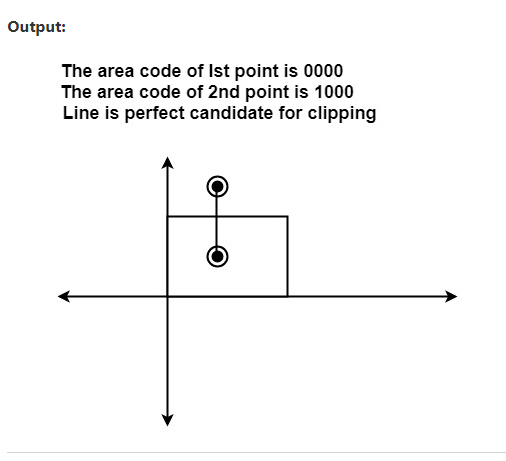
# Q2. Write a program to implement mid-point circle drawing algorithm.

1. #include <graphics.h>
2. #include <stdlib.h>
3. #include <math.h>
4. #include <stdio.h>
5. #include <conio.h>
6. #include <iostream.h>
8. **class** bresen
9. {
10. **float** x, y,a, b, r, p;
11. **public**:
12. **void** get ();
13. **void** cal ();
14. };
15. **void** main ()
16. {
17. bresen b;
18. b.get ();
19. b.cal ();
20. getch ();
21. }
22. Void bresen :: get ()
23. {
24. cout<<"ENTER CENTER AND RADIUS";
25. cout<< "ENTER (a, b)";
26. cin>>a>>b;
27. cout<<"ENTER r";
28. cin>>r;
29. }
30. **void** bresen ::cal ()
31. {
32. /\* request auto detection \*/
33. **int** gdriver = DETECT,gmode, errorcode;
34. **int** midx, midy, i;
35. /\* initialize graphics and local variables \*/
36. initgraph (&gdriver, &gmode, " ");
37. /\* read result of initialization \*/
38. errorcode = graphresult ();
39. **if** (errorcode ! = grOK)    /\*an error occurred \*/
40. {
41. printf("Graphics error: %s \n", grapherrormsg (errorcode);
42. printf ("Press any key to halt:");
43. getch ();
44. exit (1); /\* terminate with an error code \*/
45. }
46. x=0;
47. y=r;
48. putpixel (a, b+r, RED);
49. putpixel (a, b-r, RED);
50. putpixel (a-r, b, RED);
51. putpixel (a+r, b, RED);
52. p=5/4)-r;
53. **while** (x<=y)
54. {
55. If (p<0)
56. p+= (4\*x)+6;
57. **else**
58. {
59. p+=(2\*(x-y))+5;
60. y--;
61. }
62. x++;
63. putpixel (a+x, b+y, RED);
64. putpixel (a-x, b+y, RED);
65. putpixel (a+x, b-y, RED);
66. putpixel (a+x, b-y, RED);
67. putpixel (a+x, b+y, RED);
68. putpixel (a+x, b-y, RED);
69. putpixel (a-x, b+y, RED);
70. putpixel (a-x, b-y, RED);
71. }
72. }



# Q3. Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

1. #include <iostream.h>
2. #include <conio.h>
3. #include <graphics.h>
4. #include <dos.h>
5. **class** data
6. {
7. **int** gd, gmode, x, y, xmin,ymin,ymax,xmax;
8. **int** a1,a2;
9. **float** x1, y1,x2,y2,x3,y3;
10. **int** xs, ys, xe, ye;
11. **float** maxx,maxy;
12. **public**:
13. **void** getdata ();
14. **void** find ();
15. **void** clip ();
16. **void** display (**float**, **float**,**float**,**float**);
17. **void** checkonof (**int**);
18. **void** showbit (**int**);
19. };
20. **void** data :: getdata ()
21. {
22. cout<<"Enter the minimum and maximum coordinate of window (x, y) ";
23. cin >>xmin>>ymin>>xmax>>ymax;
24. cout<<"Enter the end points of the line to be clipped";
25. cin >>xs>>ys>>xe>>ye;
26. display (xs, ys, xe,ye);
27. }
28. **void** data :: display (**float**, xs, **float**, ys,**float** xe, **float** ye)
29. {
30. **int** gd=DETECT;
31. initgraph (&gd,&gmode, "");
32. maxx=getmaxx();
33. maxy=getmaxy();
34. line (maxx/2,0,maxx/2,maxy);
35. line (0, maxy/2,maxx,maxy/2);
36. rectangle (maxx/2+xmin,maxy/2-ymax,maxx/2+xmax,maxy/2-ymin);
37. line (maxx/2+xs,maxy/2-ys,maxx/2+xe,maxy/2-ye);
38. getch();
39. }
40. **void** data :: find ()
41. {
42. a1=0;
43. a2=0;
44. **if** ((ys-ymax)>0)
45. a1+=8;
46. **if** ((ymin-ys)>0)
47. a1+=4;
48. **if** ((xs-xmax)>0)
49. a1+=2;
50. **if** ((xmin-xs)>0)
51. a1+=1;
52. **if** ((ye-ymax)>0)
53. a2+=8;
54. **if** ((ymin-ye)>0)
55. a2+=4;
56. **if** ((xe-xmax)>0)
57. a2+=2;
58. **if** ((xmin-xe)>0)
59. a2+=1;
60. cout<<"\nThe area code of Ist point is ";
61. showbit (a1);
62. getch ();
63. cout <<"\nThe area code of 2nd point is ";
64. showbit (a2);
65. getch ();
66. }
67. **void** data :: showbit (**int** n)
68. {
69. **int** i,k, and;
70. **for** (i=3;i>=0;i--)
71. {
72. and =1<<i;
73. k = n?
74. k ==0?cout<<"0": cout<<"1\"";
75. }
76. }
77. **void** data ::clip()
78. {
79. **int** j=a1&a2;
80. **if** (j==0)
81. {
82. cout<<"\nLine is perfect candidate for clipping";
83. **if** (a1==0)
84. {
85. **else**
86. {
87. checkonof(a1);
88. x2=x1;y2=y1;
89. }
90. **if** (a2=0)
91. {
92. x3=xe; y3=ye;
93. }
94. **else**
95. {
96. checkonof (a2);
97. x3=x1; y3=y1;
98. }
99. xs=x2; ys=y2;xe=x3;ye=y3;
100. cout << endl;
101. display (xs,ys,xe,ye);
102. cout<<"Line after clipping";
103. getch ()
104. }
105. **else** **if** ((a1==0) && (a2=0))
106. {
107. cout <<"\n Line is in the visible region";
108. getch ();
109. }
110. }
111. **void** data :: checkonof (**int** i)
112. {
113. **int** j, k,l,m;
114. 1=i&1;
115. x1=0;y1=0;
116. **if** (1==1)
117. {
118. x1=xmin;
119. y1=ys+ ((x1-xs)/ (xe-xs))\*(ye-ys);
120. }
121. j=i&8;
122. **if** (j>0)
123. {
124. y1=ymax;
125. x1=xs+(y1-ys)/(ye-ys))\*(xe-xs);
126. }
127. k=i & 4;
128. **if** (k==1)
129. {
130. y1=ymin;
131. x1=xs+((y1-ys)/(ye-ys))\*(xe-xs);
132. }
133. m= i&2;
134. **if** (m==1)
135. {
136. x1=xmax;
137. y1=ys+ ((x1-xs)/ (xe-xs))\*(ye-ys);
138. }
139. main ()
140. {
141. data s;
142. clrscr();
143. s.getdata();
144. s.find();
145. getch();
146. closegraph ();
147. **return** ();
148. }



# Q4. Write a program to clip a polygon using Sutherland Hodgeman algorithm.

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#define round(a) ((int)(a+0.5))

int k;

float xmin,ymin,xmax,ymax,arr[20],m;

void clipl(float x1,float y1,float x2,float y2)

{

if(x2-x1)

m=(y2-y1)/(x2-x1);

else

m=100000;

if(x1 >= xmin && x2 >= xmin)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(x1 < xmin && x2 >= xmin)

{

arr[k]=xmin;

arr[k+1]=y1+m\*(xmin-x1);

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(x1 >= xmin && x2 < xmin)

{

arr[k]=xmin;

arr[k+1]=y1+m\*(xmin-x1);

k+=2;

}

}

void clipt(float x1,float y1,float x2,float y2)

{

if(y2-y1)

m=(x2-x1)/(y2-y1);

else

m=100000;

if(y1 <= ymax && y2 <= ymax)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(y1 > ymax && y2 <= ymax)

{

arr[k]=x1+m\*(ymax-y1);

arr[k+1]=ymax;

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(y1 <= ymax && y2 > ymax)

{

arr[k]=x1+m\*(ymax-y1);

arr[k+1]=ymax;

k+=2;

}

}

void clipr(float x1,float y1,float x2,float y2)

{

if(x2-x1)

m=(y2-y1)/(x2-x1);

else

m=100000;

if(x1 <= xmax && x2 <= xmax)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(x1 > xmax && x2 <= xmax)

{

arr[k]=xmax;

arr[k+1]=y1+m\*(xmax-x1);

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(x1 <= xmax && x2 > xmax)

{

arr[k]=xmax;

arr[k+1]=y1+m\*(xmax-x1);

k+=2;

}

}

void clipb(float x1,float y1,float x2,float y2)

{

if(y2-y1)

m=(x2-x1)/(y2-y1);

else

m=100000;

if(y1 >= ymin && y2 >= ymin)

{

arr[k]=x2;

arr[k+1]=y2;

k+=2;

}

if(y1 < ymin && y2 >= ymin)

{

arr[k]=x1+m\*(ymin-y1);

arr[k+1]=ymin;

arr[k+2]=x2;

arr[k+3]=y2;

k+=4;

}

if(y1 >= ymin && y2 < ymin)

{

arr[k]=x1+m\*(ymin-y1);

arr[k+1]=ymin;

k+=2;

}

}

void main()

{

int gdriver=DETECT,gmode,n,poly[20];

float xi,yi,xf,yf,polyy[20];

clrscr();

cout<<"Coordinates of rectangular clip window :\nxmin,ymin :";

cin>>xmin>>ymin;

cout<<"xmax,ymax :";

cin>>xmax>>ymax;

cout<<"\n\nPolygon to be clipped :\nNumber of sides :";

cin>>n;

cout<<"Enter the coordinates :";

for(int i=0;i < 2\*n;i++)

cin>>polyy[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

for(i=0;i < 2\*n+2;i++)

poly[i]=round(polyy[i]);

initgraph(&gdriver,&gmode,"C:\\TC\\BGI");

setcolor(RED);

rectangle(xmin,ymax,xmax,ymin);

cout<<"\t\tUNCLIPPED POLYGON";

setcolor(WHITE);

fillpoly(n,poly);

getch();

cleardevice();

k=0;

for(i=0;i < 2\*n;i+=2)

clipl(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

for(i=0;i < 2\*n;i+=2)

clipt(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

for(i=0;i < 2\*n;i+=2)

clipr(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

n=k/2;

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

polyy[i]=arr[i];

polyy[i]=polyy[0];

polyy[i+1]=polyy[1];

k=0;

for(i=0;i < 2\*n;i+=2)

clipb(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);

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

poly[i]=round(arr[i]);

if(k)

fillpoly(k/2,poly);

setcolor(RED);

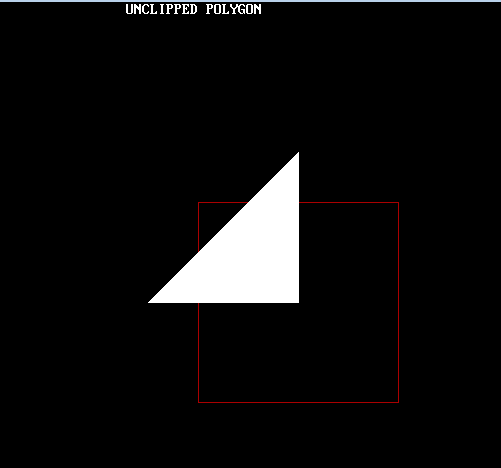
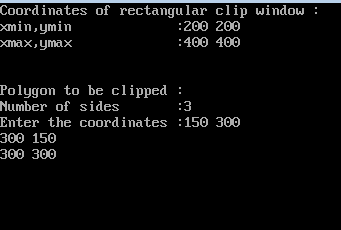
rectangle(xmin,ymax,xmax,ymin);

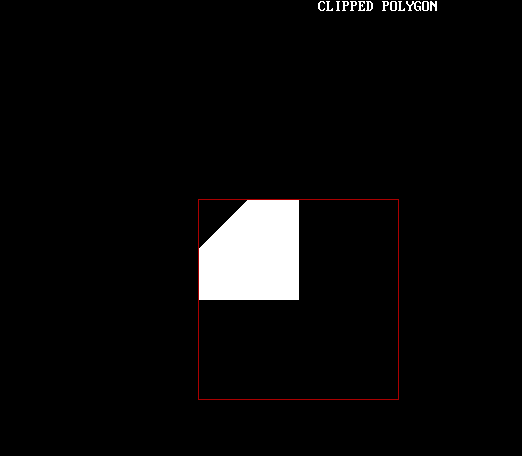
cout<<"\tCLIPPED POLYGON";

getch();

closegraph();

}





# Q5. Write a program to fill a polygon using Scan line fill algorithm.

**#include <conio.h>**

**#include <iostream>**

**#include <graphics.h>**

**#include <stdlib.h>**

**using namespace std;**

**//Declaration of class point**

**class point**

**{**

**public:**

**int x,y;**

**};**

**class poly**

**{**

**private:**

**point p[20];**

**int inter[20],x,y;**

**int v,xmin,ymin,xmax,ymax;**

**public:**

**int c;**

**void read();**

**void calcs();**

**void display();**

**void ints(float);**

**void sort(int);**

**};**

**void poly::read()**

**{**

**int i;**

**cout<<"\n\t SCAN\_FILL ALGORITHM";**

**cout<<"\n Enter the no of vertices of polygon:";**

**cin>>v;**

**if(v>2)**

**{**

**for(i=0;i<v; i++) //ACCEPT THE VERTICES**

**{**

**cout<<"\nEnter the co-ordinate no.- "<<i+1<<" : ";**

**cout<<"\n\tx"<<(i+1)<<"=";**

**cin>>p[i].x;**

**cout<<"\n\ty"<<(i+1)<<"=";**

**cin>>p[i].y;**

**}**

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

**p[i].y=p[0].y;**

**xmin=xmax=p[0].x;**

**ymin=ymax=p[0].y;**

**}**

**else**

**cout<<"\n Enter valid no. of vertices.";**

**}**

**//FUNCTION FOR FINDING**

**void poly::calcs()**

**{ //MAX,MIN**

**for(int i=0;i<v;i++)**

**{**

**if(xmin>p[i].x)**

**xmin=p[i].x;**

**if(xmax<p[i].x)**

**xmax=p[i].x;**

**if(ymin>p[i].y)**

**ymin=p[i].y;**

**if(ymax<p[i].y)**

**ymax=p[i].y;**

**}**

**}**

**//DISPLAY FUNCTION**

**void poly::display()**

**{**

**int ch1;**

**char ch='y';**

**float s,s2;**

**do**

**{**

**cout<<"\n\nMENU:";**

**cout<<"\n\n\t1 . Scan line Fill ";**

**cout<<"\n\n\t2 . Exit ";**

**cout<<"\n\nEnter your choice:";**

**cin>>ch1;**

**switch(ch1)**

**{**

**case 1:**

**s=ymin+0.01;**

**delay(100);**

**cleardevice();**

**while(s<=ymax)**

**{**

**ints(s);**

**sort(s);**

**s++;**

**}**

**break;**

**case 2:**

**exit(0);**

**}**

**cout<<"Do you want to continue?: ";**

**cin>>ch;**

**}while(ch=='y' || ch=='Y');**

**}**

**void poly::ints(float z) //DEFINE FUNCTION INTS**

**{**

**int x1,x2,y1,y2,temp;**

**c=0;**

**for(int i=0;i<v;i++)**

**{**

**x1=p[i].x;**

**y1=p[i].y;**

**x2=p[i+1].x;**

**y2=p[i+1].y;**

**if(y2<y1)**

**{**

**temp=x1;**

**x1=x2;**

**x2=temp;**

**temp=y1;**

**y1=y2;**

**y2=temp;**

**}**

**if(z<=y2&&z>=y1)**

**{**

**if((y1-y2)==0)**

**x=x1;**

**else // used to make changes in x. so that we can fill our polygon after cerain distance**

**{**

**x=((x2-x1)\*(z-y1))/(y2-y1);**

**x=x+x1;**

**}**

**if(x<=xmax && x>=xmin)**

**inter[c++]=x;**

**}**

**}**

**}**

**void poly::sort(int z) //SORT FUNCTION**

**{**

**int temp,j,i;**

**for(i=0;i<v;i++)**

**{**

**line(p[i].x,p[i].y,p[i+1].x,p[i+1].y); // used to make hollow outlines of a polygon**

**}**

**delay(100);**

**for(i=0; i<c;i+=2)**

**{**

**delay(100);**

**line(inter[i],z,inter[i+1],z); // Used to fill the polygon ....**

**}**

**}**

**int main() //START OF MAIN**

**{**

**int cl;**

**initwindow(500,600);**

**cleardevice();**

**poly x;**

**x.read();**

**x.calcs();**

**cleardevice();**

**cout<<"\n\tEnter the colour u want:(0-15)->"; //Selecting colour**

**cin>>cl;**

**setcolor(cl);**

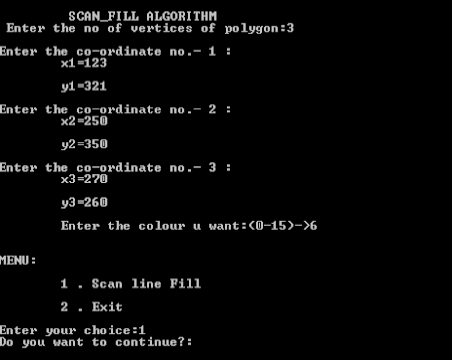
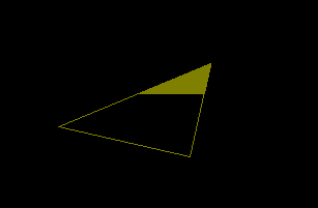
**x.display();**

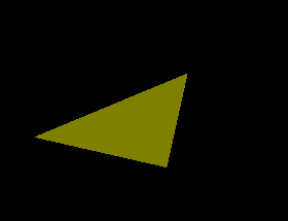
**closegraph(); //CLOSE OF GRAPH**

**getch();**

**return 0;**

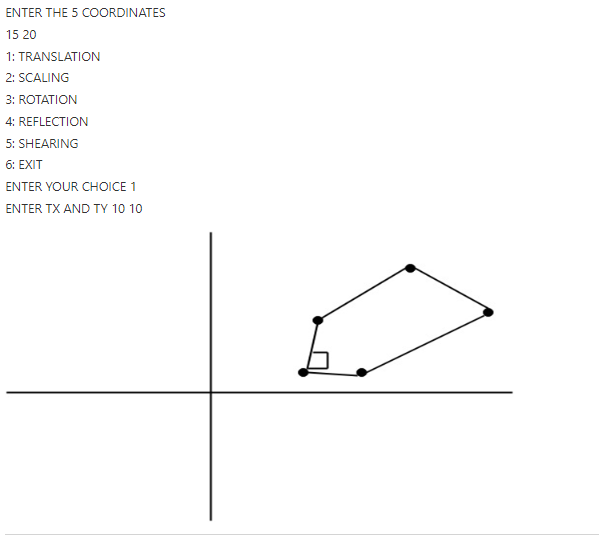
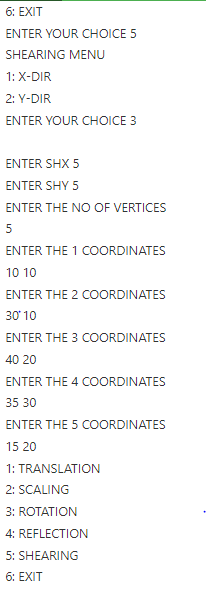
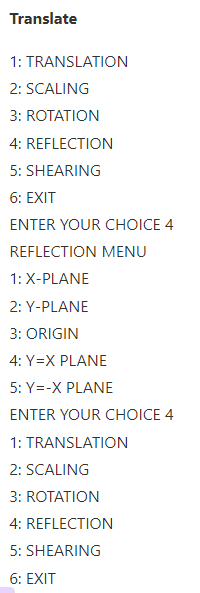
**}**

** **

****

# Q6.Write a program to apply various 2D transformations on a 2D object (use homogenous coordinates).

1. #include<iostream.h>
2. #include<conio.h>
3. #include<math.h>
4. #include<stdlib.h>
5. #include<conio.h>
6. **class** trans
7. {
8. **float** x[20],y[20],xm,ym,ref[2][2],shx,shy;
9. **int** i,j,k,n;
10. **float** sx,sy,tx,ty,ang;
11. **int** gd,gm;
12. **float** xtmp [20],ytmp[20];
13. **public**:
14. **void** takeinput();
15. **void** menu();
16. **void** graphmode();
17. **void** mapgraph();
18. **void** plotint();
19. **void** translate();
20. **void** scale();
21. **void** rotate();
22. **void** reflect();
23. **void** shear();
24. **void** plotfinal();
25. };
26. **int** ch;
27. **void** trans::takeinput()
28. {
29. cout<<"ENTER THE NO OF VERTICES\n";
30. cin>>n;
31. **for** (i=0;i<n;i++)
32. {
33. cout<<"ENTER THE "<<i+1<<"COORDINATES \n";
34. cin>>x[i]>>y[i];
35. }
36. clrscr();
37. }
38. **void** trans::menu()
39. {
40. **int** kk;
41. cout<<"\n1:TRANSLATION";
42. cout<<"\n2:SCALING";
43. cout<<"\n3:ROTATION";
44. cout<<"\n4:REFLECTION";
45. cout<<"\n5:SHEARING";
46. cout<<"\n6:EXIT";
47. cin>>ch;
48. **switch** (ch)
49. {
50. case1:
51. cout<<"\n ENTER TX AND TY";
52. cin>>tx>>ty;
53. **break**;
54. case2:
55. cout<<"\n ENTER SX AND SY";
56. cin>>sx>>sy;
57. **break**;
58. case3:
59. cout<<"\n ENTER ANGLE OF ROTATION";
60. cin>>ang;
61. **break**;
62. case4:
63. cout<<"\n REFLECTION MENU";
64. cout<<"\n 1:X-PLANE";
65. cout<<"\n 2: Y-PLANE";
66. cout<<"\n 3: ORIGIN";
67. cout<<"\n 4: Y=X PLANE";
68. cout<<"\n 5: Y=-X PLANE";
69. cout<<"\n ENTER YOUR CHOICE";
70. cin>>kk;
71. **switch** (kk)
72. {
73. case1:
74. ref [0][0] =1;
75. ref [0][1]=0;
76. ref [1][0]=0;
77. ref [1][1]=1;
78. **break**;
79. case2:
80. ref [0][0]= -1;
81. ref [0][1]=0;
82. ref [1][0]=0;
83. ref [1][1]=1;
84. **break**;
85. case3:
86. ref [0][0]=-1;
87. ref [0][1]=0;
88. ref [1][0]=0;
89. ref [1][1]=1;
90. **break**;
91. case4:
92. ref [0][0]=0;
93. ref [0][1]=1;
94. ref [1][0] =1;
95. ref [1][1]=0;
96. **break**;
97. case5:
98. ref [0][0]=0;
99. ref [0][1]=1;
100. ref [1][0]=1;
101. ref [1][1]=0;
102. **break**;
103. case5:
104. cout<< "\n SHEARING MENU";
105. cout<<"\n 1:X-DIR\n 2: Y-DIR \n 3: X-Y DIR\n ENTER YOUR               CHOICE";
106. cin>>kk;
107. **switch** (kk)
108. {
109. case1:
110. cout<<"\n ENTER SHX";
111. cin>> shx;
112. ref[0][0] =1;
113. ref [0][1]=0;
114. ref [1][0]=shx;
115. ref [1][1]=1;
116. **break**;
117. case2:
118. cout<< "\n ENTER SHY";
119. cin>>shy;
120. ref [0][0]=1;
121. ref [0][1]=shy;
122. ref [1][0]=0;
123. ref [1][1] =1;
124. **break**;
125. case3:
126. cout<<"\n ENTER SHX";
127. cin >> shx;
128. cout<<"\n ENTER SHY";
129. cin>> shy;
130. ref [0][0] =1;
131. ref [0][1] =shy;
132. ref [1][0] =shx;
133. ref [1][1] =1;
134. **break**;
135. }
136. **break**;
137. }
138. }
139. **void** trans::graphmode()
140. {
141. gd=DETECT;
142. initgraph (&gd, &gm, "");
143. }
144. **void** trans::mapgraph()
145. {
146. xm=getmaxx ()/2;
147. ym=getmaxy ()/2;
148. line (xm,0,xm,2\*ym);
149. line (0,ym,2 \* xm,ym);
150. }
151. **void** trans::plotint()
152. {
153. **for**(i=0;i<n-1;i++)
154. {
155. circle (x[i] +xm,-y[i]+ym,2)
156. circle x [n-1]+xm,(-y[n-1]+ym),2;
157. line (x[i]+xm,(-y[i]+ym),x[i+1]+xm,(-y[i+1]+ym));
158. }
159. line (x[n-1]+xm,(-y[n-1]+ym,)x[0]+xm,(-y[0]+ym));
160. }
161. **void** trans::translate()
162. {
163. **for**(i=0;i<n;i++)
164. {
165. xtmp[i]=x[i]+tx;
166. ytmp[i]=y[i]+ty;
167. }
168. }
169. **void** trans::plotfinal()
170. {
171. **for** (i=0;i<n-1;i++)
172. {
173. circle (xtmp[i]+xm, (-ytmp[i]+ym,2);
174. circle (xtmp[n-1]+xm,(-ytmp[n-1]+ym),2);
175. line (xtmp[i]+xm,(-ytmp[i]+ym),xtmp[i+1]+xm,(-ytmp[i+1]+ym));
176. }
177. line (xtmp[n-1]+xm,(-ytmp[n-1]+ym),xtmp[0]+xm,(-ytmp[0]+ym));
178. }
179. **void** trans::scale()
180. {
181. **float** s [2][2],mxy[7][2],rxy[7][2];
182. s [0][0]=sx;
183. s [0][1]=0;
184. s [1][0]=0;
185. s [1][1]=sy;
186. tx=-x[0];
187. ty=-y[0];
188. translate ();
189. k=0;
190. **for**(i=0;i<n;i++)
191. {
192. j=0;
193. mxy[i][j]=xtmp[k];
194. mxy[i][j+1]=ytmp[k];
195. k++;
196. }
197. **for** (i=0;i<n;i++)
198. {
199. **for**(j=0;j<2;j++)
200. {
201. rxy[i][j]=0;
202. **for**(k=0;k<2;k++)
203. {
204. rxy[i][j]+=mxy[i][k]\*s[k][j];
205. }
206. }
207. }
208. j=0;
209. k=0;
210. **for**(i=0;i<n;i++)
211. {
212. j=0;
213. x[k]=rxy[i][j];
214. y[k]=rxy[i][j+1];
215. k++;
216. }
217. tx=-tx;
218. ty=-ty;
219. translate();
220. }
221. **void** trans::rotate()
222. {
223. **float** r[2][2],mxy[7][2],rxy[7][2],tmp;
224. tmp=22/7;
225. tmp=(tmp\*ang)/180;
226. r[0][0]=cos(tmp);
227. r[0][1]=sin(tmp);
228. r[1][0]=cos(tmp);
229. r[1][1]=sy;
230. tx=-x[0];
231. ty=-y[0];
232. translate ();
233. k=0;
234. **for** (i=0;i<n;i++)
235. {
236. j=0;
237. mxy[i][j]=xtmp[k];
238. mxy[i][j+1]=ytmp[k];
239. k++;
240. }
241. **for** (i=0;i<n;i++)
242. {
243. **for** (j=0;j<2;j++)
244. {
245. rxy[i][j]=0;
246. **for** (k=0;k<2;k++)
247. {
248. rxy[i][j]+=mxy[i][k]\*r[k][j];
249. }
250. }
251. }
252. j=0;
253. k=0;
254. **for**(i=0;i<n;i++)
255. {
256. j=0;
257. x[k]=rxy[i][j];
258. y[k]=rxy[i][j+1];
259. k++;
260. }
261. tx=-tx;
262. ty=-ty;
263. translate();
264. }
265. **void** trans::reflect()
266. {
267. **float** mxy[7][2],rxy[7][2],tmp;
268. tx=0;
269. ty=0;
270. translate();
271. k=0;
272. **for**(i=0;i<n;i++)
273. {
274. j=0;
275. mxy[i][j]=xtmp[k];
276. mxy[i][j+1]=ytmp[k];
277. k++;
278. }
279. **for**(i=0;i<n;i++)
280. {
281. **for**(j=0;j<2;j++)
282. {
283. rxy[i][j]=0;
284. **for**(k=0;k<2;k++)
285. {
286. rxy[i][j]|+=mxy[i][k]\*r[k][j];
287. }
288. }
289. }
290. j=0;
291. k=0;
292. **for**(i=0;i<n;i++)
293. {
294. j=0;
295. x[k]=rxy[i][j];
296. y[k]=rxy[i][j+1];
297. k++;
298. }
299. tx=-tx;
300. ty=-ty;
301. translate();
302. }
303. **void** trans::shear()
304. {
305. **float** mxy[7][2],rxy[7][2],tmp;
306. tx=0;
307. ty=0;
308. translate ();
309. k=0;
310. **for**(i=0;i<n;i++)
311. {
312. j=0;
313. mxy[i][j]=xtmp[k];
314. mxy[i][j+1]=ytmp[k];
315. k++;
316. }
317. **for**(i=0;i<n;i++)
318. {
319. **for**(j=0;j<2;j++)
320. {
321. rxy[i][j]=0;
322. **for** (k=0;k<2;k++)
323. {
324. rxy[i][j]|+=mxy[i][k]\*r[k][j];
325. }
326. }
327. }
328. j=0;
329. k=0;
330. **for**(i=0;i<n;i++)
331. {
332. j=0;
333. x[k]=rxy[i][j];
334. y[k]=rxy[i][j+1];
335. k++;
336. }
337. tx=-tx;
338. ty=-ty;
339. translate ();
340. }
341. **void** main()
342. {
343. clrscr ();
344. trans t1;
345. t1.takeinput ();
346. t1.menu ();
347. t1.graphmode ();
348. t1.mapgraph ();
349. t1.plotint ();
350. **switch** (ch)
351. {
352. case1:
353. t1.translate ();
354. **break**;
355. case2:
356. t1.scale ();
357. **break** ();
358. case3:
359. t1.rotate ();
360. **break**;
361. case4:
362. t1.reflect ();
363. **break**;
364. case5:
365. t1.shear ();
366. **break**;
367. case6:
368. exit ();
369. }
370. getch ();
371. t1.plotfinal ();
372. getch ();
373. closegraph ();
374. }

****

# Q7. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

#include<stdio.h>  
#include<conio.h>  
#include<graphics.h>  
#include<math.h>  
int maxx,maxy,midx,midy;  
  
void axis()  
{  
  getch();  
  cleardevice();  
  line(midx,0,midx,maxy);  
  line(0,midy,maxx,midy);  
}  
void main()  
{  
  int 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();  
}

# Q8. Write a program to draw Hermite/Bezier curve.

#include<graphics.h>

#include<math.h>

#include<conio.h>

#include<stdio.h>

void main()

{

int x[4],y[4],i;

double put\_x,put\_y,t;

int gr=DETECT,gm;

initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");

printf("\n\*\*\*\*\*\* Bezier Curve \*\*\*\*\*\*\*\*\*\*\*");

printf("\n Please enter x and y coordinates ");

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

{

scanf("%d%d",&x[i],&y[i]);

putpixel(x[i],y[i],3); // Control Points

}

for(t=0.0;t<=1.0;t=t+0.001) // t always lies between 0 and 1

{

put\_x = pow(1-t,3)\*x[0] + 3\*t\*pow(1-t,2)\*x[1] + 3\*t\*t\*(1-t)\*x[2] + pow(t,3)\*x[3]; // Formula to draw curve

put\_y = pow(1-t,3)\*y[0] + 3\*t\*pow(1-t,2)\*y[1] + 3\*t\*t\*(1-t)\*y[2] + pow(t,3)\*y[3];

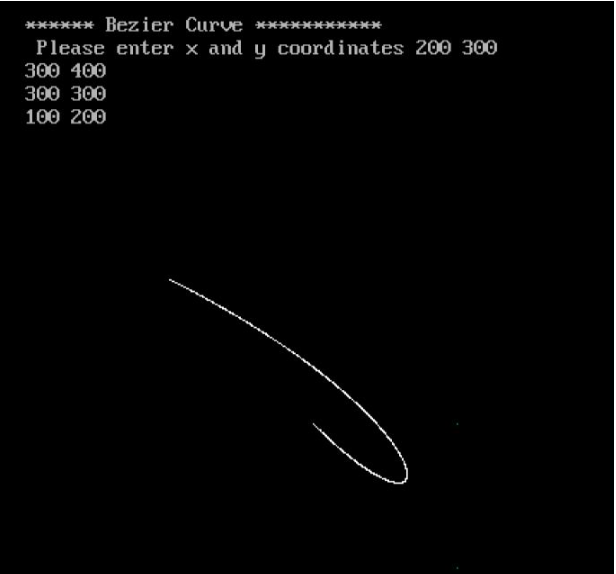
putpixel(put\_x,put\_y, WHITE); // putting pixel

}

getch();

closegraph();

}

****