COMPUTER GRAPHICS

PRACTICAL FILE

TANIA CHAUHAN

19/78121

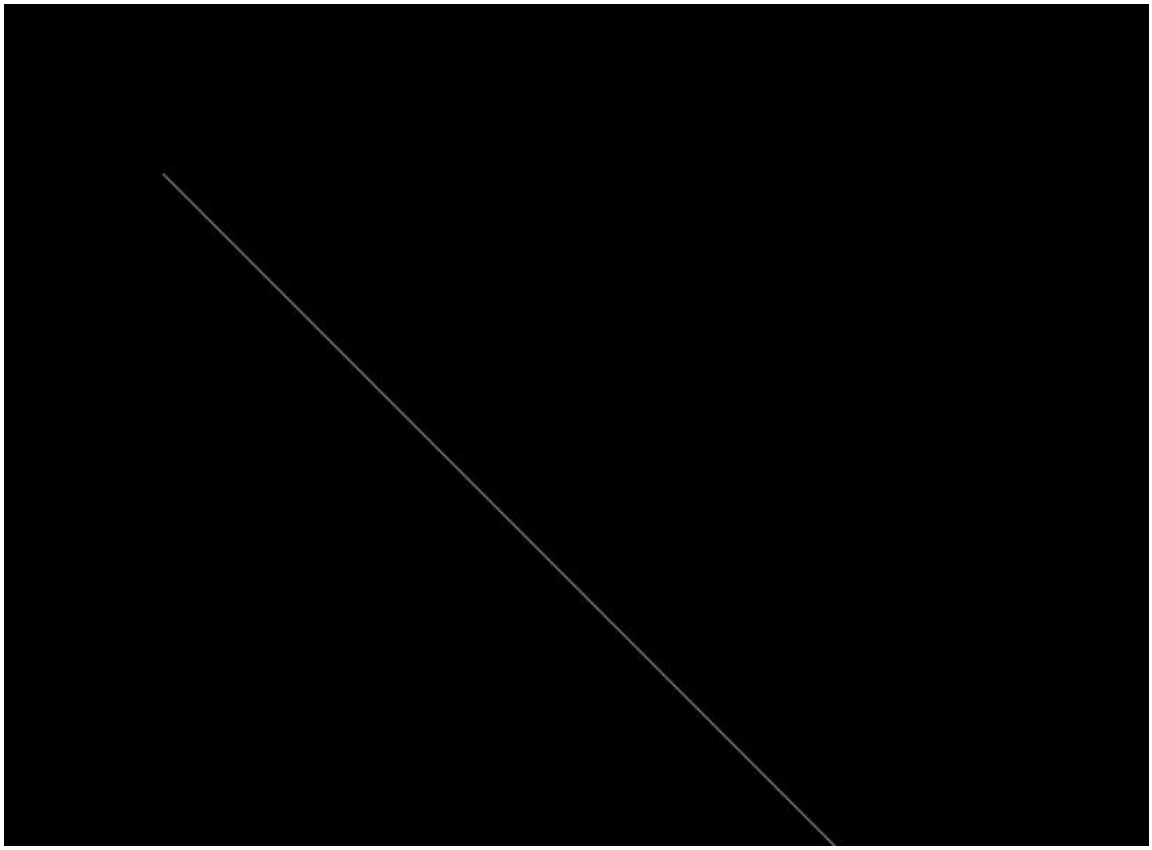
BSc(H) COMPUTER SCIENCE

Q1. Write a program to implement Bresenham's line drawing algorithm.

```
#include <graphics.h>
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;
void drawline(int x1,int y1,int x2,int y2)
{
   int dx,dy,p,x,y;
   dx=x2-x1;
   dy=y2-y1;
   x=x1;
   y=y1;
   p=2*dy-dx;
while(x1<x2)
   {
      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;
   }
}
```

```
int main()
{
  int gdriver=DETECT,gmode,error,x1,y1,x2,y2;
  initgraph(&gdriver,&gmode,NULL);
    cout<<"Enter first coordinate:";
    cin>>x1>>y1;
    cout<<"Enter second coordinate:";
    cin>>x2>>y2;
    drawline(x1,y1,x2,y2);
    getch();
    return 0;
}
```

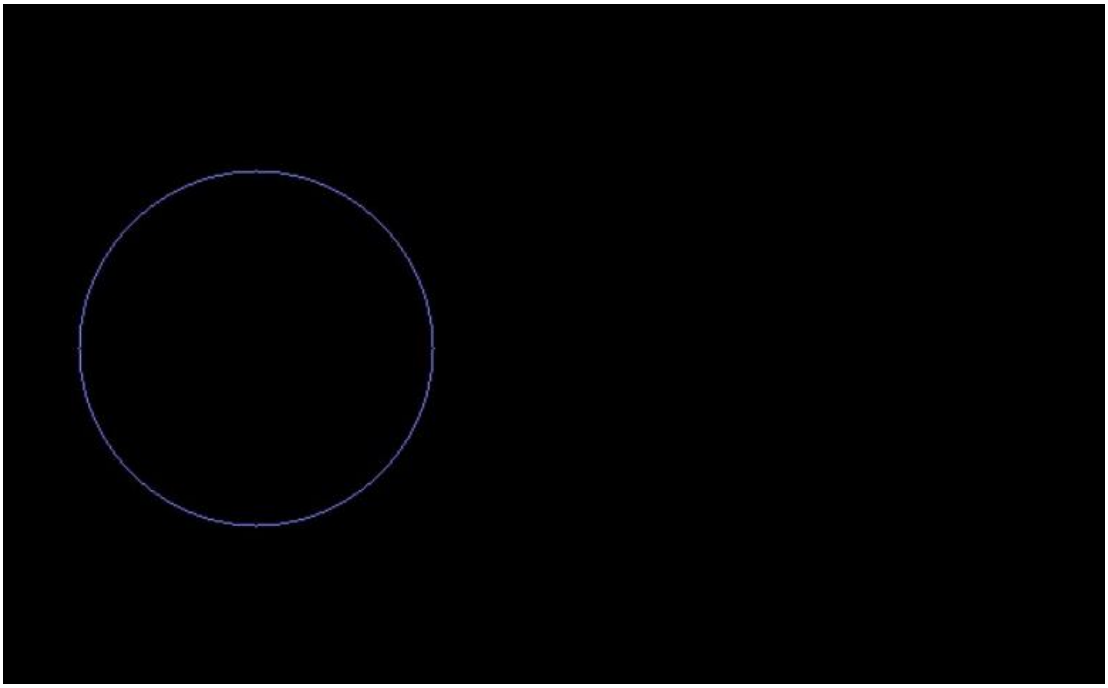
```
Enter first coordinate:90
100
Enter second coordinate:150
160
```



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

```
#include <graphics.h>
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;
void drawcircle(int x1,int y1,int rad)
{
   int x=rad;
   int y=0;
   int err=0;
   while (x>=y)
    {
      putpixel(x1+x,y1+y,9);
      putpixel(x1+y,y1+x,9);
      putpixel(x1-y,y1+x,9);
      putpixel(x1-x,y1+y,9);
      putpixel(x1-x,y1-y,9);
      putpixel(x1-y,y1-x,9);
      putpixel(x1+y,y1-x,9);
      putpixel(x1+x,y1-y,9);
   if (err<=0)
    {
      y+=1;
      err+=2*y+1;
    }

   if(err>0)
    {
      x-=1;
      err-=2*x+1;
    }
   }
}
int main()
{
   int gdriver=DETECT,gmode,error,x,y,rad;
   initgraph(&gdriver,&gmode,NULL);
   cout<<"Enter radius of circle:";
   cin>>rad;
   cout<<"Enter co-ordinates of center(x,y): ";
   cin>>x>>y;
   drawcircle(x,y,rad);
    getch();
   return 0;
}
```

```
Enter radius of circle:100
Enter co-ordinates of center(x,y): 150
200
PS C:\Users\Sameer\OneDrive\Documents\graphics>
```



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

```
#include <iostream>
#include <stdio.h>
#include <graphics.h>
#include <vector>
using namespace std;
typedef unsigned int outcode;
enum _boolean {_false, _true};
enum {
  _top = 0x1,
  _bottom = 0x2,
```

```
  _right = 0x4,
  _left = 0x8
};
outcode compoutcode(double _x, double _y, double _xmin, double _xmax, double _ymin, double
_ymax) {
  outcode code = 0;
  if (_y > _ymax)
    code |= _top;
  else if (_y < _ymin)
    code |= _bottom;
  else if (_x > _xmax)
    code |= _right;
  else if (_x < _xmin)
    code |= _left;
  return code;
}
void cohen_sutherland_line_clip(double _x0, double _y0, double _x1, double _y1, double _xmin,
double _xmax, double _ymin, double _ymax) {
  outcode outcode0, outcode1, outcodeOut;
  _boolean accept = _false, done = _false;
  outcode0 = compoutcode(_x0, _y0, _xmin, _xmax, _ymin, _ymax);
  outcode1 = compoutcode(_x1, _y1, _xmin, _xmax, _ymin, _ymax);
  do {
    if (!(outcode0 | outcode1)) {
      accept = _true;
      done = _true;
    } else if (outcode0 & outcode1) {
      done = _true;
    } else {
      double x, y;
      outcodeOut = outcode0 ? outcode0 : outcode1;
      if (outcodeOut & _top) {
        x = _x0 + (_x1 - _x0) * (_ymax - _y0) / (_y1 - _y0);
        y = _ymax;
      } else if (outcodeOut & _bottom) {
        x = _x0 + (_x1 - _x0) * (_ymin - _y0) / (_y1 - _y0);
        y = _ymin;
      } else if (outcodeOut & _right) {
        y = _y0 + (_y1 - _y0) * (_xmax - _x0) / (_x1 - _x0);
        x = _xmax;
      } else {
        y = _y0 + (_y1 - _y0) * (_xmin - _x0) / (_x1 - _x0);
        x = _xmin;
      }
      if (outcodeOut == outcode0) {
        _x0 = x;
        _y0 = y;
        outcode0 = compoutcode(_x0, _y0, _xmin, _xmax, _ymin, _ymax);
```

```
    } else {
      _x1 = x;
      _y1 = y;
      outcode1 = compoutcode(_x1, _y1, _xmin, _xmax, _ymin, _ymax);
    }
   }
  } while (done == _false);

  if (accept) {
   line(_x0, _y0, _x1, _y1);
   cout << "\nThe clipped co-ordinates of line are:"
      << "\n(x0, y0) : (" << _x0 << ", " << _y0 << ")"
      << "\n(x1, y1) : (" << _x1 << ", " << _y1 << ")"
      << endl;
  }
}

int main() {
  cout << "\n============ COHEN AND SUTHERLAND ALGORITHM ============\n";
  int x0, y0, x1, y1;
  int xmin, xmax, ymin, ymax;
  int lines_count;
  vector<vector<int> > lines;
  vector<int> point;
  int gd = DETECT, gm;
  initgraph(&gd, &gm, NULL);
  setbkcolor(RED);
  cout << "\n\nEnter the co-ordinates of the rectangle:";
  cout << "\nXmin : ";
  cin >> xmin;
  cout << "Xmax : ";
  cin >> xmax;
  cout << "Ymin : ";
  cin >> ymin;
  cout << "Ymax : ";
  cin >> ymax;
  rectangle(xmin, ymin, xmax, ymax);
  cout << "\nEnter the no. of lines: ";
  cin >> lines_count;
  for (int i = 0; i < lines_count; i++) {
   cout << "\nEnter the co-ordinates of the line " << i+1 << " :";
   cout << "\nx0 : ";
   cin >> x0;
   cout << "y0 : ";
   cin >> y0;
   cout << "x1 : ";
   cin >> x1;
   cout << "y1 : ";
```

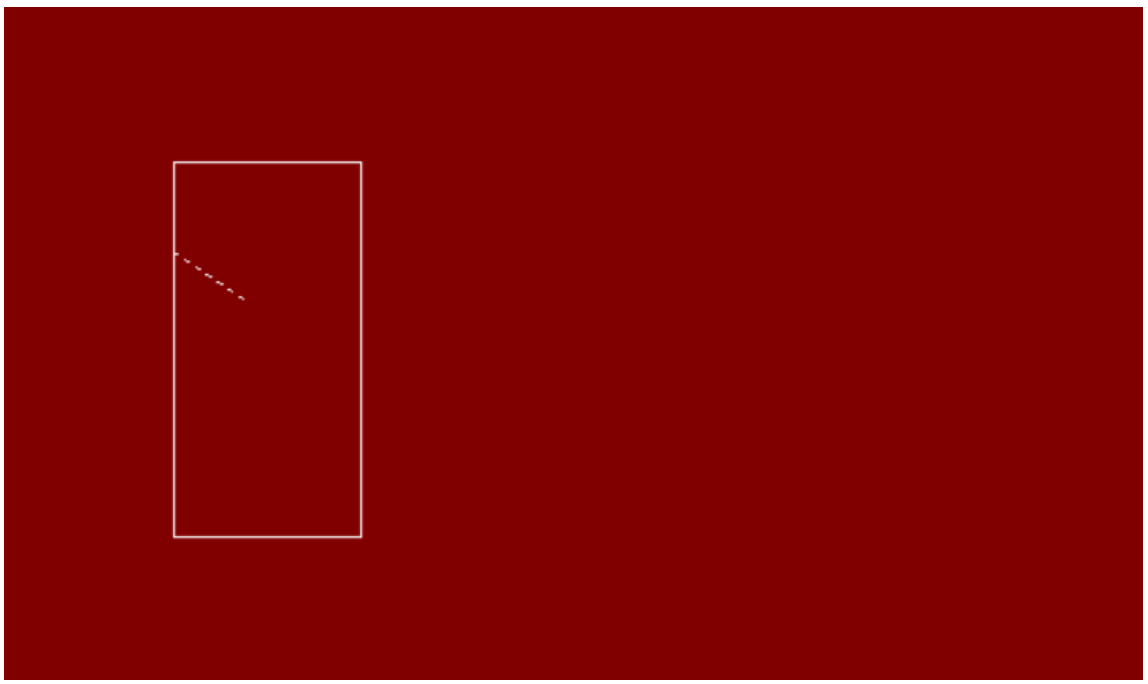
```
    cin >> y1;
    point.push_back(x0);
    point.push_back(y0);
    point.push_back(x1);
    point.push_back(y1);
    lines.push_back(point);
    point.clear();
  }
  cout << "\nThe line before clipping...\n";
  for (int i = 0; i < lines_count; i++) {
    x0 = lines[i][0];
    y0 = lines[i][1];
    x1 = lines[i][2];
    y1 = lines[i][3];
    line(x0, y0, x1, y1);
  }
  delay(3000);
  cleardevice();
  delay(200);
  cout << "\nThe line after clipping...\n";
  rectangle(xmin, ymin, xmax, ymax);
  setlinestyle(DOTTED_LINE, 1, 1);

  for (int i = 0; i < lines_count; i++) {
    x0 = lines[i][0];
    y0 = lines[i][1];
    x1 = lines[i][2];
    y1 = lines[i][3];
    cohen_sutherland_line_clip(x0, y0, x1, y1, xmin, xmax, ymin, ymax);
  }
  getch();
  return 0;
}
```

```
=========== COHEN AND SUTHERLAND ALGORITHM ===========


Enter the co-ordinates of the rectangle:
Xmin : 100
Xmax : 200
Ymin : 100
Ymax : 300

Enter the no. of lines: 2
```

```
Ymax : 300

Enter the no. of lines: 2

Enter the co-ordinates of the line 1 :
x0 : 80
y0 : 135
x1 : 140
y1 : 175

Enter the co-ordinates of the line 2 :
x0 : 70
y0 : 140
```

```
Enter the co-ordinates of the line 2 :
x0 : 70
y0 : 140
x1 : 90
y1 : 250

The line before clipping...

The line after clipping...

The clipped co-ordinates of line are:
(x0, y0) : (100, 148.333)
```

```
The line before clipping...

The line after clipping...

The clipped co-ordinates of line are:
(x0, y0) : (100, 148.333)
(x1, y1) : (140, 175)
```



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

```
#include<iostream>
#include<graphics.h>
using namespace std;
const int MAX_POINTS = 20;
int draw_points[20];
void getPolyPoints(int array[][2], int size, int result[]) {
  int i, k;
  for(i = 0, k = 0; i < size; i++, k += 2) {
    result[k] = array[i][0];
    result[k + 1] = array[i][1];
  }

  result[k] = array[0][0];
  result[k + 1] = array[0][1];
}
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(int poly_points[][2], 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++) {
       int k = (i+1) % poly_size;
       int ix = poly_points[i][0], iy = poly_points[i][1];
       int kx = poly_points[k][0], ky = poly_points[k][1];
       int i_pos = (x2-x1) * (iy-y1) - (y2-y1) * (ix-x1);
       int k_pos = (x2-x1) * (ky-y1) - (y2-y1) * (kx-x1);
       if (i_pos < 0  && k_pos < 0)
       {
          new_points[new_poly_size][0] = kx;
          new_points[new_poly_size][1] = ky;
          new_poly_size++;
       }
       else if (i_pos >= 0  && k_pos < 0)
       {
          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++;
        }
        else if (i_pos < 0  && k_pos >= 0)
        {
            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++;
        }
        else
        {
          //NO POINTS ARE ADDED
        }
    }
    poly_size = new_poly_size;
    for(int i = 0; i < poly_size; i++) {
      poly_points[i][0] = new_points[i][0];
      poly_points[i][1] = new_points[i][1];
    }
}

void suthHodgClip(int poly_points[][2], int poly_size, int clipper_points[][2], int clipper_size) {
    for(int i = 0; i < clipper_size; i++) {
      int k = (i + 1) % clipper_size;
      clip(poly_points, poly_size, clipper_points[i][0], clipper_points[i][1], clipper_points[k][0],
clipper_points[k][1]);
    }
    for(int i = 0; i < poly_size; i++) {
      cout<<"("<<poly_points[i][0]<<","<<poly_points[i][1]<<")";
      cout<<endl;
    }
    int gd = DETECT, gm;
    char data[] = "C://turboc3//bgi";
    initgraph(&gd, &gm, data);
    getPolyPoints(clipper_points, clipper_size, draw_points);
    setcolor(RED);
    drawpoly(clipper_size + 1, draw_points);
    getPolyPoints(poly_points, poly_size, draw_points);
    delay(1000);
    setcolor(DARKGRAY);
    drawpoly(poly_size + 1, draw_points);
}

int main() {
    int poly_size;
    int poly_points[20][2];
    int clipper_size;
    int clipper_points[20][2];
```

```
 cout<<"\n================================================================================\n";
 cout<<"\t\tSUTHERLAND HODGEMAN POLYGON CLIPPING ALGORITHM";
 cout<<"\n================================================================================";
 cout<<"\nEnter the number of vertices clipping window has: ";
 cin>>clipper_size;
 cout<<"Enter the coordinates of the clipping window:-\n";
 for(int i = 0; i < clipper_size; i++) {
  cout<<"x"<<i<<": ";
  cin>>clipper_points[i][0];
  cout<<"y"<<i<<": ";
  cin>>clipper_points[i][1];
  cout<<endl;
 }
 cout<<"\n\nEnter the number of vertices polygon has: ";
 cin>>poly_size;
 cout<<"Enter the coordinates of the clipping window:-\n";
 for(int i = 0; i < poly_size; i++) {
  cout<<"x"<<i<<": ";
  cin>>poly_points[i][0];
  cout<<"y"<<i<<": ";
  cin>>poly_points[i][1];
  cout<<endl;
 }

 cout<<"\nCLIPPING IS BEING PERFORMED...";
 int gd = DETECT, gm;
 char data[] = "C://turboc3//bgi";

 initgraph(&gd, &gm, data);

 getPolyPoints(clipper_points, clipper_size, draw_points);
 setcolor(RED);
 drawpoly(clipper_size + 1, draw_points);
 getPolyPoints(poly_points, poly_size, draw_points);
 delay(1000);
 setcolor(DARKGRAY);
 drawpoly(poly_size + 1, draw_points);
 suthHodgClip(poly_points, poly_size, clipper_points, clipper_size);
 cout<<"\nCLIPPING IS DONE!";
 getch();
 closegraph();
 return 1;
}
```

```
Enter the coordinates of the clipping window:-
x0: 54
y0: 43

x1: 23
y1: 32

x2: 24
y2: 42


CLIPPING IS BEING PERFORMED...
CLIPPING IS DONE!
```

Q6. Write a program to apply various 2D transformation on a 2D object.

```
#include<iostream>
#include<graphics.h>
#include<cmath>
#include<cstdlib>
using namespace std;
void disp(int n,float c[][3])
{
float xmax,ymax;
int i;
xmax=getmaxx();
ymax=getmaxy();
xmax=xmax/2;
ymax=ymax/2;
i=0;
while(i<n-1)
{
line(xmax+c[i][0],ymax-c[i][1],xmax+c[i+1][0],ymax-c[i+1][1]);
i++;
}
i=n-1;
line(xmax+c[i][0],ymax-c[i][1],xmax+c[0][0],ymax-c[0][1]);
```

```
setcolor(15);
line(0,ymax,xmax*2,ymax);
line(xmax,0,xmax,ymax * 2);
setcolor(WHITE);
}
void mul(int n,float b[][3],float c[][3],float a[][3])
{
int i,j,k;
for (i=0;i<n;i++)
for (j=0;j<3;j++)
a[i][j]=0;
for (i=0;i<n;i++)
for (j=0;j<3;j++)
for (k=0;k<3;k++)
{
a[i][j]=a[i][j]+(c[i][k]*b[k][j]);
}
}
void translation(int n,float c[][3],float tx,float ty)
{
int i;
for(i=0;i<n;i++)
{
c[i][0]=c[i][0]+tx;
c[i][1]=c[i][1]+ty;
}
}
void scaling(int n,float c[][3],float sx,float sy)
{
float b[10][3],a[10][3];
int i=0,j;
for (i=0;i<3;i++)
for (j=0;j<3;j++)
b[i][j]=0;
b[0][0]=sx;
b[1][1]=sy;
b[2][2]=1;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);
}
void rotation(int n,float c[][3],float ra)
{
int i=0,j;
float b[10][3],xp,yp,a[10][3];
xp=c[0][0];
yp=c[0][1];
for(i=0;i<3;i++)
```

```
for (j=0;j<3;j++)
b[i][j]=0;
b[0][0]=b[1][1]=cos(ra*3.14/180);
b[0][1]=sin(ra*3.14/180);
b[1][0]=-sin(ra*3.14/180);
b[2][0]=(-xp*cos(ra*3.14/180))+(yp*sin(ra*3.14/180))+xp;
b[2][1]=(-xp*sin(ra*3.14/180))-(yp *cos(ra*3.14/180))+yp;
b[2][2]=1;
mul(n,b,c,a);
setcolor(15);
disp(n,a);
}
void refthrx(int n,float c[][3])
{
int i=0,j;
float a[10][3],b[10][3];
for (i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=1;
b[0][1]=0;
b[0][2]=0;
b[1][0]=0;
b[1][1]=-1;
b[1][2]=0;
b[2][0]=0;
b[2][1]=0;
b[2][2]=1;
mul(n,b,c,a);
setcolor(3);
disp(n,a);
}
void refthry(int n,float c[][3])
{
int i=0,j;
float b[10][3],a[10][3];
for (i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=-1;
b[0][1]=0;
b[0][2]=0;
b[1][0]=0;
b[1][1]=1;
b[1][2]=0;
b[2][0]=0;
b[2][1]=0;
b[2][2]=1;
```

```
mul(n,b,c,a);
setcolor(3);
disp(n,a);
}
void refthrxeqtoy(int n,float c[][3])
{
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=0;
b[0][1]=1;
b[0][2]=0;
b[1][0]=1;
b[1][1]=0;
b[1][2]=0;
b[2][0]=0;
b[2][1]=0;
b[2][2]=1;
mul(n,b,c,a);
setcolor(3);
disp(n,a);
}
void refthrxnegy(int n,float c[][3])
{
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=0;
b[0][1]=-1;
b[0][2]=0;
b[1][0]=-1;
b[1][1]=0;
b[1][2]=0;
b[2][0]=0;
b[2][1]=0;
b[2][2]=1;
mul(n,b,c,a);
setcolor(3);
disp(n,a);
}
void refaboutorigin(int n,float c[][3])
{
int i=0,j;
float b[10][3],a[10][3];
```

```
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=-1;
b[1][1]=-1;
b[2][2]=1;
mul(n,b,c,a);
setcolor(3);
disp(n,a);
}
void xshearwithy(int n,float c[][3],float shx)
{
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=b[1][1]=b[2][2]=1;
b[1][0]=shx;
mul(n,b,c,a);
setcolor(3);
disp(n,a);
}
void shearing(int n,float c[][3])
{
float b[10][3],sh,a[10][3];
int i=0,ch,j;
cleardevice();
cout<<"\n\t* * * MENU * * *";
cout<<"\n\t1) X Shearing";
cout<<"\n\t2) Y Shearing";
cout<<"\n\t1) EXIT";
cout<<"\n\tEnter your Choice: ";
cin>> ch;
if(ch==3)
return;
cout<<"\n\tEnter the value for Shearing";
cin>>sh;
cleardevice();
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
for (i=0;i<3;i++)
b[i][i]=1;
switch(ch)
{
case 1:
disp(n,c);
```

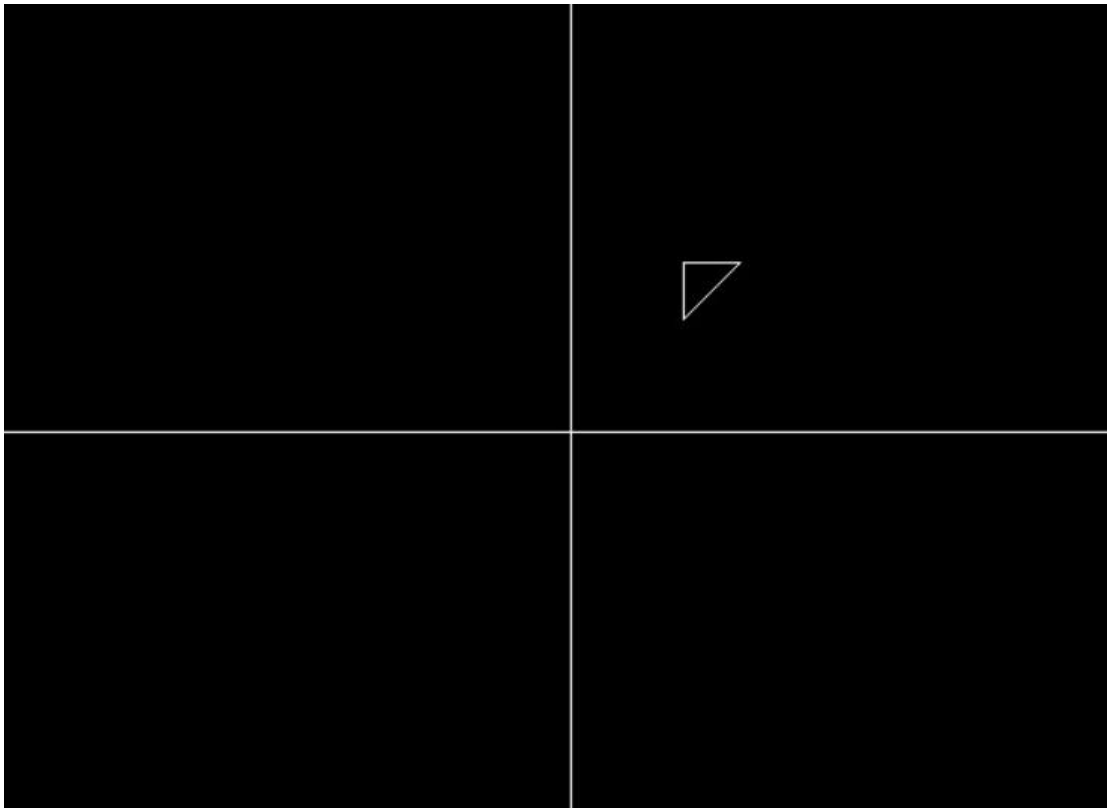
```
b[1][0]=sh;
break;
case 2:
disp(n,c);
b[0][1]=sh;
break;
case 3:
break;
default:
cout <<"\n\tINVALID CHOICE !!!";
break;
}
mul(n,b,c,a);
setcolor(RED);
disp(n,a);
}
int main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,NULL);
int i,j,k,ch,n,ch2,shx,shy,yref,xref,ch3;
float c[10][3],tx,ty,sx,sy,ra;
cout<<"Enter the number of vertices :";
cin>>n;
for(i=0;i<n;i++)
{
cout<<"Enter the coordinates of the vertex :",i + 1;
cin>>c[i][0]>>c[i][1];
c[i][2]=1;
}
do
{
cleardevice();
cout<<"\n\t\t * * * MENU * * *";
cout<<"\n\t 1) TRANSLATION";
cout<<"\n\t 2) SCALING";
cout<<"\n\t 3) ROTATION";
cout<<"\n\t 4) REFLECTION";
cout<<"\n\t 5) SHEARING";
cout<<"\n\t 6) Exit";
cout<<"Enter your Choice :";
cin>>ch;
switch(ch)
{
case 1:
cout<<"\n\tEnter Translation factor for X and Y axis:\t";
cin>>tx>>ty;
cleardevice();
```

```
setcolor(15);
disp(n,c);
translation(n,c,tx,ty);
setcolor(15);
disp(n,c);
getch();
break;
case 2:
cout<<"\n\tEnter scaling Factor for X and Y axis :\t";
cin>>sx>>sy;
cleardevice();
setcolor(15);
disp(n,c);
scaling(n,c,sx,sy);
getch();
break;
case 3:
cout<<"\n\n\tEnter the angle of Rotation";
cin>>ra;
cleardevice();
setcolor(15);
rotation(n,c,ra);
getch();
break;
case 4:
cout<<"\n1. Ref thru x axis";
cout<<"\n2. Ref thru y axis";
cout<<"\n3. Ref thru x=y axis";
cout<<"\n4. ref thru x=-y axis";
cout<<"\n5. Ref about origin";
cout<<"\n Enter your choice: ";
cin>>ch2;
switch(ch2)
{
case 1:
setcolor(15);
disp(n,c);
refthrx(n,c);
break;
case 2:
setcolor(15);
disp(n,c);
refthry(n,c);
break;
case 3:
setcolor(15);
disp(n,c);
refthrxeqtoy(n,c);
```

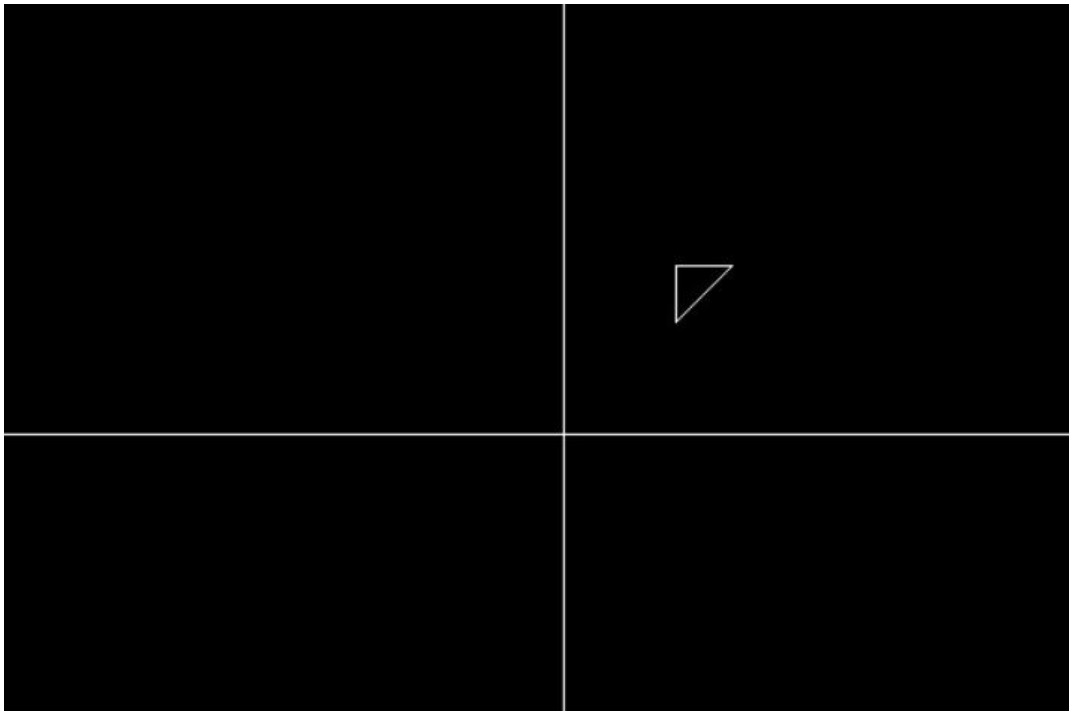
```
break;
case 4:
setcolor(15);
disp(n,c);
refthrxnegy(n,c);
break;
case 5:
setcolor(15);
disp(n,c);
refaboutorigin(n,c);
break;
}
break;
case 5:
setcolor(15);
disp(n,c);
shearing(n,c);
getch();
break;
case 6:
exit(0);
break;
default:
cout<<"\n\t Invalid Choice ! !";
break;
}
} while(ch!=4);
getch();
closegraph();
}
```

```
Enter the number of vertices :3
Enter the coordinates of the vertex :60
60
Enter the coordinates of the vertex :60
90
Enter the coordinates of the vertex :90
90
```

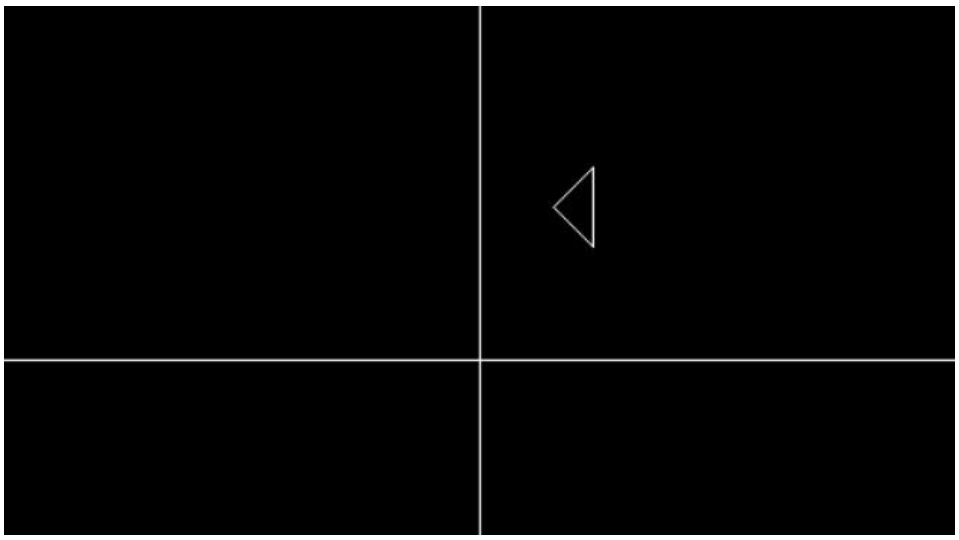
```
        * * * MENU * * *
  1) TRANSLATION
  2) SCALING
  3) ROTATION
  4) REFLECTION
  5) SHEARING
  6) ExitEnter your Choice :1

 Enter Translation factor for X and Y axis:      x
```



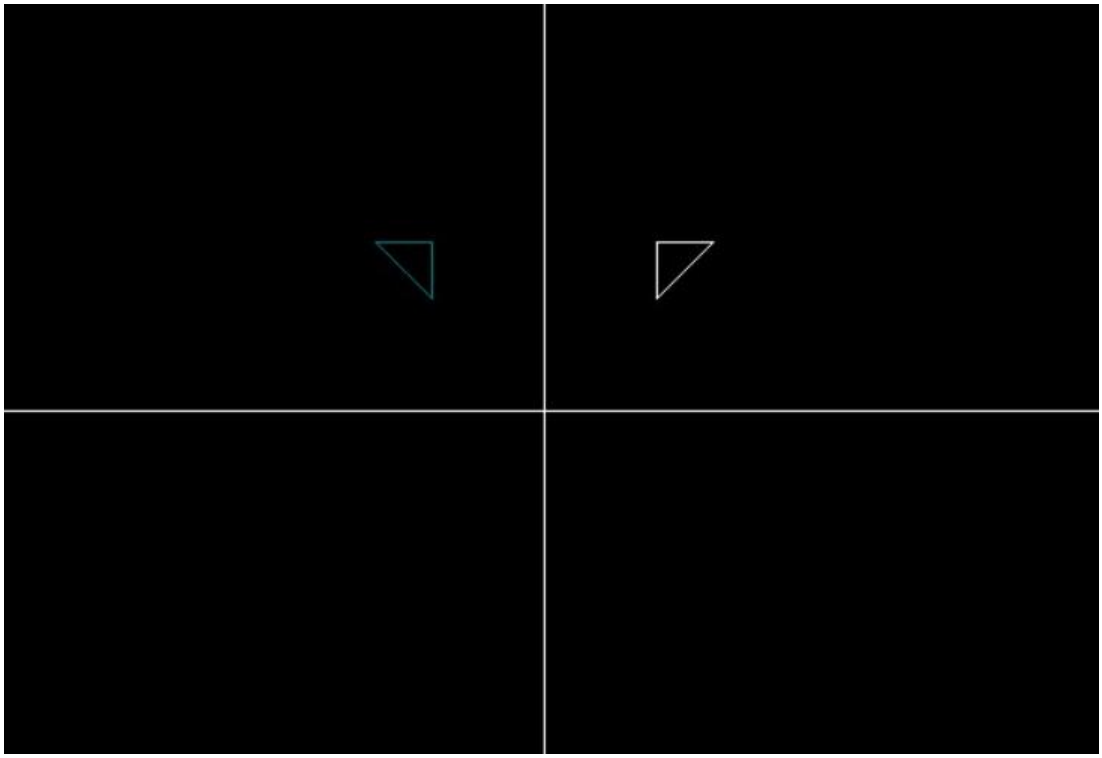
```
        * * * MENU * * *
  1) TRANSLATION
  2) SCALING
  3) ROTATION
  4) REFLECTION
  5) SHEARING
  6) ExitEnter your Choice :2

 Enter scaling Factor for X and Y axis : y
```



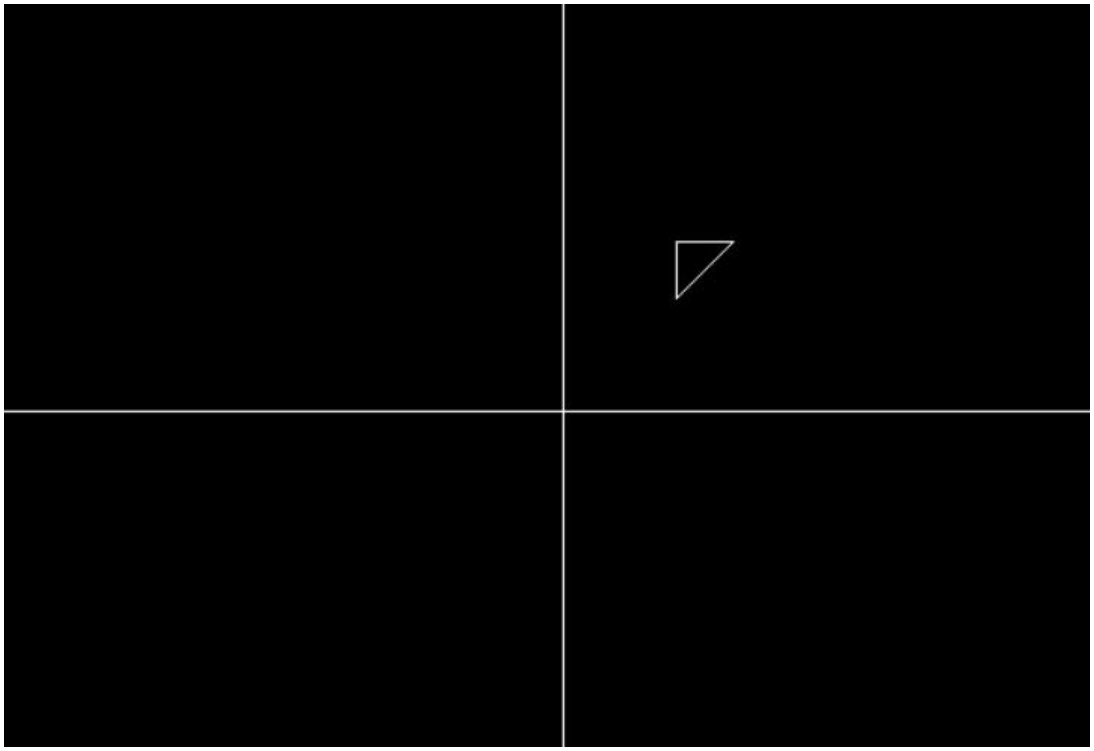
```
        * * * MENU * * *
  1) TRANSLATION
  2) SCALING
  3) ROTATION
  4) REFLECTION
  5) SHEARING
  6) ExitEnter your Choice :3


Enter the angle of Rotation45
```



```
PROBLEMS    OUTPUT    TERMINAL    DEBUG CONSOLE

        1) TRANSLATION
        2) SCALING
        3) ROTATION
        4) REFLECTION
        5) SHEARING
        6) ExitEnter your Choice :4

1. Ref thru x axis
2. Ref thru y axis
3. Ref thru x=y axis
4. ref thru x=-y axis
5. Ref about origin
 Enter your choice: 2
```



```
        3) ROTATION
        4) REFLECTION
        5) SHEARING
        6) ExitEnter your Choice :5

       * * * MENU * * *
       1) X Shearing
       2) Y Shearing
       1) EXIT
       Enter your Choice: 1

       Enter the value for Shearing20
```



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

```
#include <iostream>
#include <graphics.h>
#include <cctype>
#include <math.h>
using namespace std;
int cube[8][4];
double transform[4][4];
int cube_t[8][4];
int result[8][4];
void multi_matrix()
{
   for (int i = 0; i < 8; i++)
   {
      for (int j = 0; j < 4; j++)
      {
         cube_t[i][j] = 0;
         for (int k = 0; k < 4; k++)
         {
            cube_t[i][j] += cube[i][k] * transform[k][j];
         }
      }
```

```
    }
}
void display(int max_x, int max_y)
{
    int i;
    for (i = 0; i < 3; i++)
    {
        line(max_x + result[i][0], max_y - result[i][1], max_x + result[i + 1][0], max_y - result[i + 1][1]);
    }
    line(max_x + result[3][0], max_y - result[3][1], max_x + result[0][0], max_y - result[0][1]);
    for (i = 4; i < 7; i++)
    {
        line(max_x + result[i][0], max_y - result[i][1], max_x + result[i + 1][0], max_y - result[i + 1][1]);
    }
    line(max_x + result[7][0], max_y - result[7][1], max_x + result[4][0], max_y - result[4][1]);
    line(max_x + result[0][0], max_y - result[0][1], max_x + result[4][0], max_y - result[4][1]);
    line(max_x + result[1][0], max_y - result[1][1], max_x + result[5][0], max_y - result[5][1]);
    line(max_x + result[3][0], max_y - result[3][1], max_x + result[7][0], max_y - result[7][1]);
    line(max_x + result[2][0], max_y - result[2][1], max_x + result[6][0], max_y - result[6][1]);
}

// Scaling down the co-ordinate system to the middle of the screen
void scale_down(int max_x, int max_y)
{
    max_x = getmaxx() / 2;
    max_y = getmaxy() / 2;

    setcolor(WHITE);
    line(0, max_y, max_x * 2, max_y);
    line(max_x, 0, max_x, max_y * 2);
}

// Perform projection; if axis = 1 => Projection about x-axis || if axis = 2 => Projection about y-axis
|| if axis = 3 => Projection about z-axis
void projection_ortho()
{
    int px = 1, py = 1, pz = 0;

    // Projection matrix
    int p_t[4][4];
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j)
            {
                if (i == 0)
                {
```

```
                p_t[i][j] = px;
            }
            else if (i == 1)
            {
                p_t[i][j] = py;
            }
            else if (i == 2)
            {
                p_t[i][j] = pz;
            }
            else if (i == 3)
            {
                p_t[i][j] = 1;
            }
        }
        else
        {
            p_t[i][j] = 0;
        }
    }
}

// matrix multiplication
for (int i = 0; i < 8; i++)
{
    for (int j = 0; j < 4; j++)
    {
        result[i][j] = 0;
        for (int k = 0; k < 4; k++)
        {
            result[i][j] += cube_t[i][k] * p_t[k][j];
        }
    }
}

int gd = DETECT, gm;
// initwindow(1000, 1000);
initgraph(&gd, &gm, (char *)"");
int max_x = getmaxx() / 2, max_y = getmaxy() / 2;
scale_down(max_x, max_y);
setcolor(LIGHTBLUE);

// draw the transformed figure
display(max_x, max_y);
}

// Perform scaling on the cube
void scaling(float fx, float fy, float fz)
```

```
{
    // Update the transformation matrix
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j)
            {
                if (i == 0)
                {
                    transform[i][j] = fx;
                }
                else if (i == 1)
                {
                    transform[i][j] = fy;
                }
                else if (i == 2)
                {
                    transform[i][j] = fz;
                }
                else if (i == 3)
                {
                    transform[i][j] = 1;
                }
            }
            else
            {
                transform[i][j] = 0;
            }
        }
    }

    multi_matrix();
}

// Perform overall scaling
void scaling_overall(float f)
{
    // Update the transformation matrix
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j && i == 3)
            {
                transform[i][j] = f;
            }
            else if (i == j && i != 3)
```

```
            {
                transform[i][j] = 1;
            }
            else
            {
                transform[i][j] = 0;
            }
        }
    }

    multi_matrix();

    // normalizing
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            cube_t[i][j] /= f;
        }
    }
}

// Perform rotation about the x-axis; flag = 0 => clockwise || flag = 1 => anti-clockwise
void rotationX(int theta, int flag)
{
    if (flag = 0)
    {
        theta = -theta;
    }

    // Update the transformation matrix
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j && (i == 0 || i == 3))
            {
                transform[i][j] = 1;
            }
            else if ((i == 1 && j == 1) || (i == 2 && j == 2))
            {
                transform[i][j] = cos((double)(theta * 0.0174533));
            }
            else if (i == 2 && j == 1)
            {
                transform[i][j] = -sin((double)(theta * 0.0174533));
            }
            else if (i == 1 && j == 2)
```

```
        {
            transform[i][j] = sin((double)(theta * 0.0174533));
        }
        else
        {
            transform[i][j] == 0;
        }
    }
  }

  multi_matrix();
}

// Perform rotation about the y-axis; flag = 0 => clockwise || flag = 1 => anti-clockwise
void rotationZ(int alpha, int flag)
{
  if (flag = 0)
  {
    alpha = -alpha;
  }

  // Update the transformation matrix
  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 4; j++)
    {
      if (i == j && (i == 2 || i == 3))
      {
        transform[i][j] = 1;
      }
      else if ((i == 0 && j == 0) || (i == 1 && j == 1))
      {
        transform[i][j] = cos((double)(alpha * 0.0174533));
      }
      else if (i == 0 && j == 1)
      {
        transform[i][j] = sin((double)(alpha * 0.0174533));
      }
      else if (i == 1 && j == 0)
      {
        transform[i][j] = -sin((double)(alpha * 0.0174533));
      }
      else
      {
        transform[i][j] = 0;
      }
    }
  }
```

```
    multi_matrix();
}

// Perform rotation about the z-axis; flag = 0 => clockwise || flag = 1 => anti-clockwise
void rotationY(int phi, int flag)
{
    if (flag = 0)
    {
        phi = -phi;
    }

    // Update the transformation matrix
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j && (i == 1 || i == 3))
            {
                transform[i][j] = 1;
            }
            else if ((i == 0 && j == 0) || (i == 2 && j == 2))
            {
                transform[i][j] = cos((double)(phi * 0.0174533));
            }
            else if (i == 2 && j == 0)
            {
                transform[i][j] = sin((double)(phi * 0.0174533));
            }
            else if (i == 0 && j == 2)
            {
                transform[i][j] = -sin((double)(phi * 0.0174533));
            }
            else
            {
                transform[i][j] = 0;
            }
        }
    }
}

// Perform reflection relative to the XY plane
void reflectionXY()
{
    // Update the transformation matrix
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
```

```
        {
            if (i == j)
            {
                if (i == 2)
                {
                    transform[i][j] = -1;
                }
                else
                {
                    transform[i][j] = 1;
                }
            }
            else
            {
                transform[i][j] = 0;
            }
        }
    }

    multi_matrix();
}

// Perform reflection relative to the YZ plane
void reflectionYZ()
{
    // Update the transformation matrix
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j)
            {
                if (i == 0)
                {
                    transform[i][j] = -1;
                }
                else
                {
                    transform[i][j] = 1;
                }
            }
            else
            {
                transform[i][j] = 0;
            }
        }
    }
```

```
    multi_matrix();
}

// Perform reflection relative to the XZ plane
void reflectionXZ()
{
    // Update the transformation matrix
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j)
            {
                if (i == 1)
                {
                    transform[i][j] = -1;
                }
                else
                {
                    transform[i][j] = 1;
                }
            }
            else
            {
                transform[i][j] = 0;
            }
        }
    }

    multi_matrix();
}

// Perform shearing
void shearing(float fx1, float fx2, float fy1, float fy2, float fz1, float fz2)
{
    // Update the transformation matrix
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (i == j)
            {
                transform[i][j] = 1;
            }
            else if (i == 0)
            {
                if (j == 1)
                {
```

```
                transform[i][j] = fx1;
            }
            else if (j == 2)
            {
                transform[i][j] = fx2;
            }
        }
        else if (i == 1)
        {
            if (j == 0)
            {
                transform[i][j] = fy1;
            }
            else if (j == 2)
            {
                transform[i][j] = fy2;
            }
        }
        else if (i == 2)
        {
            if (j == 0)
            {
                transform[i][j] = fz1;
            }
            else if (j == 1)
            {
                transform[i][j] = fz2;
            }
        }
        else
        {
            transform[i][j] = 0;
        }
    }
  }

  multi_matrix();
}

// Perform translation
void translation(float tx, float ty, float tz)
{
  // Update the transformation matrix
  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 4; j++)
    {
      if (i == j)
```

```
            {
                transform[i][j] = 1;
            }
            else if (i == 2)
            {
                if (j == 0)
                {
                    transform[i][j] = tx;
                }
                else if (j == 1)
                {
                    transform[i][j] = ty;
                }
                else if (j == 2)
                {
                    transform[i][j] = tz;
                }
            }
            else
            {
                transform[i][j] = 0;
            }
        }
    }

    multi_matrix();
}

int main(int argc, char const *argv[])
{
    char ch;
    int op, op_pro;

    cout << "\n\n=============================================";
    cout << "\nPERFORM 3D TRANSFORMATIONS ON A CUBE";
    cout << "\n=============================================";
    cout << "\n\nEnter the co-ordinates of cube:-\n";
    for (int i = 0; i < 8; i++)
    {
        cout << "Row " << i + 1 << " : ";
        for (int j = 0; j < 4; j++)
        {
            cin >> cube[i][j];
        }
    }
    cube[0][3] = 1;
    cube[1][3] = 1;
    cube[2][3] = 1;
```

```
cube[3][3] = 1;
cube[4][3] = 1;
cube[5][3] = 1;
cube[6][3] = 1;
cube[7][3] = 1;

do
{
   cout << "\n\n----------------------------";
   cout << "\n\tMAIN MENU";
   cout << "\n----------------------------";
   cout << "\n1. Local Scaling";
   cout << "\n2. Overall Scaling";
   cout << "\n3. Rotation about x-axis (CLOCKWISE)";
   cout << "\n4. Rotation about y-axis (CLOCKWISE)";
   cout << "\n5. Rotation about z-axis (CLOCKWISE)";
   cout << "\n6. Rotation about x-axis (ANTI-CLOCKWISE)";
   cout << "\n7. Rotation about y-axis (ANTI-CLOCKWISE)";
   cout << "\n8. Rotation about z-axis (ANTI-CLOCKWISE)";
   cout << "\n9. Reflection relative to XY plane";
   cout << "\n10. Reflection relative to YZ plane";
   cout << "\n11. Reflection relative to XZ plane";
   cout << "\n12. Shearing";
   cout << "\n13. Translation";
   cout << "\nEnter choice: ";
   cin >> op;

   switch (op)
   {
   case 1:
   {
      float fx, fy, fz;
      cout << "\nYOU ARE PERFORMING LOCAL SCALING\n";
      cout << "Enter the scaling factors:-\n";
      cout << "fx: ";
      cin >> fx;
      cout << "fy: ";
      cin >> fy;
      cout << "fz: ";
      cin >> fz;
      cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
      cout << "Select the type of projection you want to perform.";
      cout << "\n\t1. Orthographic Projection";
      cout << "\nEnter choice: ";
      cin >> op_pro;
      switch (op_pro)
      {
      case 1:
```

```
        {
            scaling(fx, fy, fz);
            projection_ortho();
        }
        }
        cout << "Scaling with factors " << fx << ", " << fy << ", " << fz << " in the x, y and z component
respectively has been done.";
    }
    break;
    case 2:
    {
        float factor;
        cout << "\nYOU ARE PERFORMING OVERALL SCALING\n";
        cout << "Enter overall scaling factor: ";
        cin >> factor;
        cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
        cout << "Select the type of projection you want to perform.";
        cout << "\n\t1. Orthographic Projection";
        cout << "\nEnter choice: ";
        cin >> op_pro;
        switch (op_pro)
        {
        case 1:
        {
            scaling_overall(factor);
            projection_ortho();
        }
        }
        cout << "Overall scaling with a factor " << factor << " has been done.";
    }
    break;
    case 3:
    {
        int theta;
        cout << "\nYOU ARE PERFORMING ROTATION ABOUT X-AXIS (CLOCKWISE)\n";
        cout << "Enter the angle by which you want to rotate the cube: ";
        cin >> theta;
        cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
        cout << "Select the type of projection you want to perform.";
        cout << "\n\t1. Orthographic Projection";
        cout << "\nEnter choice: ";
        cin >> op_pro;
        switch (op_pro)
        {
        case 1:
        {
            rotationX(theta, 0);
            projection_ortho();
```

```
    }
    }
    cout << "Rotation about x-axis in the clockwise direction has been done.";
}
break;
case 4:
{
    int phi;
    cout << "\nYOU ARE PERFORMING ROTATION ABOUT Y-AXIS (CLOCKWISE)\n";
    cout << "Enter the angle by which you want to rotate the cube: ";
    cin >> phi;
    cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
    cout << "Select the type of projection you want to perform.";
    cout << "\n\t1. Orthographic Projection";
    cout << "\nEnter choice: ";
    cin >> op_pro;
    switch (op_pro)
    {
    case 1:
    {
        rotationY(phi, 0);
        projection_ortho();
    }
    }
    cout << "Rotation about y-axis in the clockwise direction has been done.";
}
break;
case 5:
{
    int alpha;
    cout << "\nYOU ARE PERFORMING ROTATION ABOUT Z-AXIS (CLOCKWISE)\n";
    cout << "Enter the angle by which you want to rotate the cube: ";
    cin >> alpha;
    cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
    cout << "Select the type of projection you want to perform.";
    cout << "\n\t1. Orthographic Projection";
    cout << "\nEnter choice: ";
    cin >> op_pro;
    switch (op_pro)
    {
    case 1:
    {
        rotationZ(alpha, 0);
        projection_ortho();
    }
    }
    cout << "Rotation about z-axis in the clockwise direction has been done.";
}
```

```
    break;
    case 6:
    {
       int theta;
       cout << "\nYOU ARE PERFORMING ROTATION ABOUT X-AXIS (ANTI-CLOCKWISE)\n";
       cout << "Enter the angle by which you want to rotate the cube: ";
       cin >> theta;
       cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
       cout << "Select the type of projection you want to perform.";
       cout << "\n\t1. Orthographic Projection";
       cout << "\nEnter choice: ";
       cin >> op_pro;
       switch (op_pro)
       {
       case 1:
       {
          rotationX(theta, 1);
          projection_ortho();
       }
       }
       cout << "Rotation about x-axis in the anti-clockwise direction has been done.";
    }
    break;
    case 7:
    {
       int phi;
       cout << "\nYOU ARE PERFORMING ROTATION ABOUT Y-AXIS (ANTI-CLOCKWISE)";
       cout << "\nEnter the angle by which you want to rotate the cube: ";
       cin >> phi;
       cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
       cout << "Select the type of projection you want to perform.";
       cout << "\n\t1. Orthographic Projection";
       cout << "\nEnter choice: ";
       cin >> op_pro;
       switch (op_pro)
       {
       case 1:
       {
          rotationY(phi, 1);
          projection_ortho();
       }
       }
       cout << "Rotation about y-axis in the anti-clockwise direction has been done.";
    }
    break;
    case 8:
    {
       int alpha;
```

```
        cout << "\nYOU ARE PERFORMING ROTATION ABOUT Z-AXIS (ANTI-CLOCKWISE)\n";
        cout << "Enter the angle by which you want to rotate the cube: ";
        cin >> alpha;
        cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
        cout << "Select the type of projection you want to perform.";
        cout << "\n\t1. Orthographic Projection";
        cout << "\nEnter choice: ";
        cin >> op_pro;
        switch (op_pro)
        {
        case 1:
        {
            rotationZ(alpha, 1);
            projection_ortho();
        }
        }
        cout << "Rotation about z-axis in the anti-clockwise direction has been done.";
    }
    break;
    case 9:
    {
        cout << "\nYOU ARE PERFORMING REFLECTION RELATIVE TO XY PLANE\n";
        cout << "Reflection relative to the XY plane has been done.";
        cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
        cout << "Select the type of projection you want to perform.";
        cout << "\n\t1. Orthographic Projection";
        cout << "\nEnter choice: ";
        cin >> op_pro;
        switch (op_pro)
        {
        case 1:
        {
            reflectionXY();
            projection_ortho();
        }
        }
    }
    break;
    case 10:
    {
        cout << "\nYOU ARE PERFORMING REFLECTION RELATIVE TO YZ PLANE\n";
        cout << "Reflection relative to the YZ plane has been done.";
        cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
        cout << "Select the type of projection you want to perform.";
        cout << "\n\t1. Orthographic Projection";
        cout << "\nEnter choice: ";
        cin >> op_pro;
        switch (op_pro)
```

```
        {
        case 1:
        {
            reflectionYZ();
            projection_ortho();
        }
        }
    }
    break;
    case 11:
    {
        cout << "\nYOU ARE PERFORMING REFLECTION RELATIVE TO THE XZ PLANE\n";
        cout << "Reflection relative to the XZ plane has been done.";
        cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
        cout << "Select the type of projection you want to perform.";
        cout << "\n\t1. Orthographic Projection";
        cout << "\nEnter choice: ";
        cin >> op_pro;
        switch (op_pro)
        {
        case 1:
        {
            reflectionXZ();
            projection_ortho();
        }
        }
    }
    break;
    case 12:
    {
        int fx1, fx2, fy1, fy2, fz1, fz2;
        cout << "\nYOU ARE PERFORMING SHEARING\n";
        cout << "Enter the shearing factors:-\n";
        cout << "fx1: ";
        cin >> fx1;
        cout << "fx2: ";
        cin >> fx2;
        cout << "fy1: ";
        cin >> fy1;
        cout << "fy2: ";
        cin >> fy2;
        cout << "fz1: ";
        cin >> fz1;
        cout << "fz2: ";
        cin >> fz2;
        cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
        cout << "Select the type of projection you want to perform.";
        cout << "\n\t1. Orthographic Projection";
```

```
        cout << "\nEnter choice: ";
        cin >> op_pro;
        switch (op_pro)
        {
        case 1:
        {
            shearing(fx1, fx2, fy1, fy2, fz1, fz2);
            projection_ortho();
        }
        }
        cout << "Shearing with factors " << fx1 << ", " << fx2 << ", " << fy1 << ", " << fy2 << ", " << fz1 << ",
" << fz2 << " has been done.";
    }
    break;
    case 13:
    {
        float tx, ty, tz;
        cout << "\nYOU ARE PERFORMING TRANSLATION\n";
        cout << "Enter the translation factors in the x, y and z-direction respectively:-\n";
        cout << "tx: ";
        cin >> tx;
        cout << "ty: ";
        cin >> ty;
        cout << "tz: ";
        cin >> tz;
        cout << "\nTO DISPLAY THE TRANSFORMED FIGURE\n";
        cout << "Select the type of projection you want to perform.";
        cout << "\n\t1. Orthographic Projection";
        cout << "\nEnter choice: ";
        cin >> op_pro;
        switch (op_pro)
        {
        case 1:
        {
            translation(tx, ty, tz);
            projection_ortho();
        }
        }
        cout << "Translation with factors " << tx << ", " << ty << ", " << tz << " has been done in the
direction of x, y and z-axis respectively.";
    }
    break;
    default:
        cout << "\nPlease enter a valid option.";
    };

    cout << "\nWant to return back to menu? (y/Y - \"Yes\", any other key - \"No\") : ";
    cin >> ch;
```

```
  } while (toupper(ch) == 'Y');

  closegraph();
  return 0;
}
```

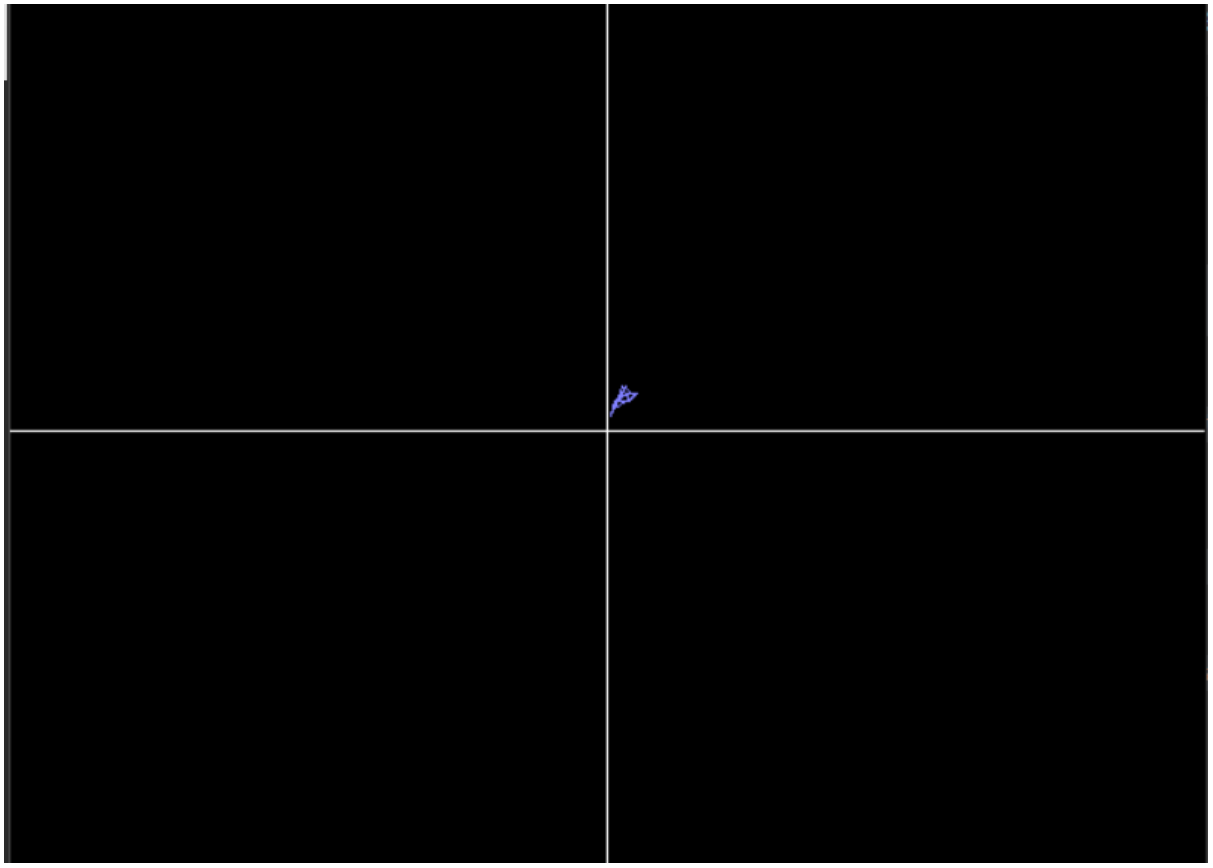
```
========================================
PERFORM 3D TRANSFORMATIONS ON A CUBE
========================================

Enter the co-ordinates of cube:-
Row 1 : 1
2
3
4
Row 2 : 5
6
7
8
Row 3 : 2
3
5
6
Row 4 : 4
6
8
9
Row 5 : 2
4
3
1
Row 6 : 6
5
7
8
Row 7 : 8
5
2
1
Row 8 : 6
4
1
3
```

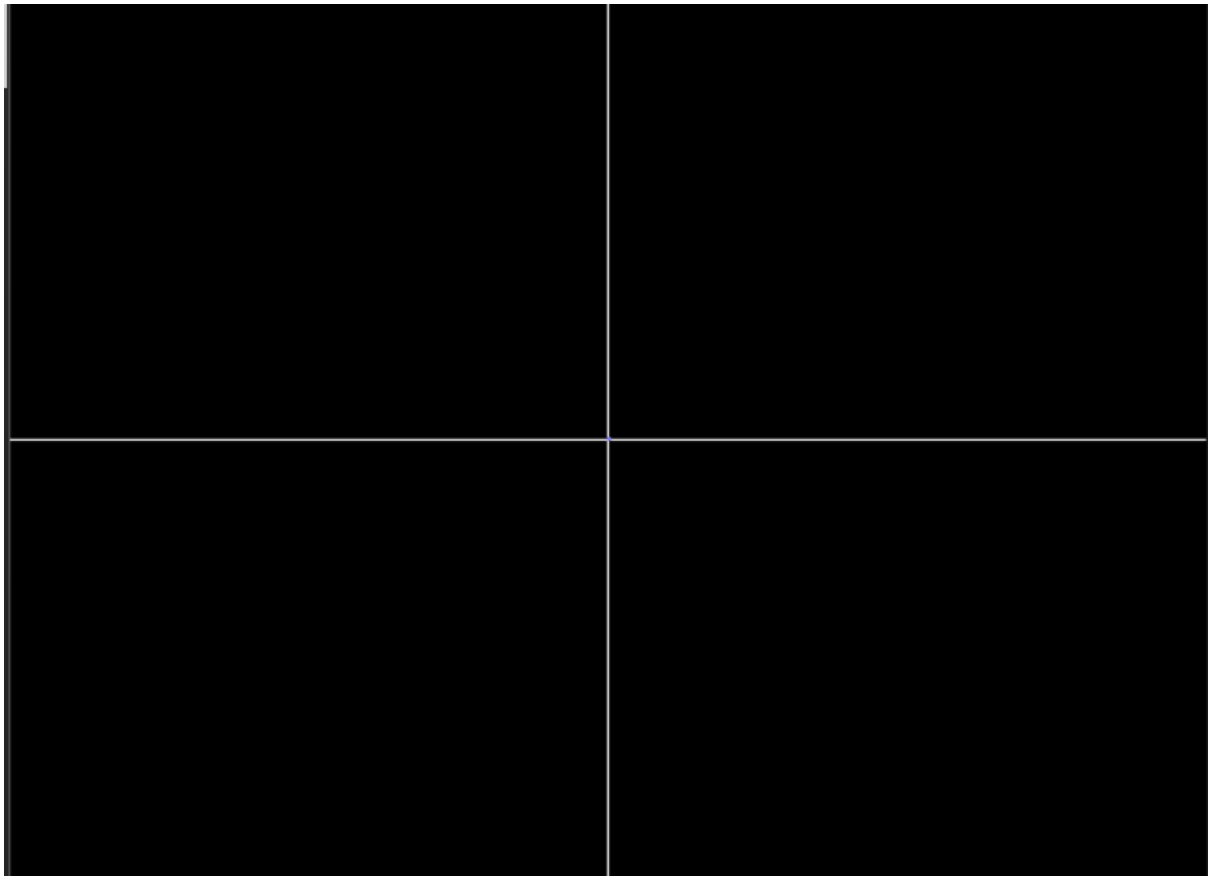
```
---------------------------
        MAIN MENU
---------------------------
1. Local Scaling
2. Overall Scaling
3. Rotation about x-axis (CLOCKWISE)
4. Rotation about y-axis (CLOCKWISE)
5. Rotation about z-axis (CLOCKWISE)
6. Rotation about x-axis (ANTI-CLOCKWISE)
7. Rotation about y-axis (ANTI-CLOCKWISE)
8. Rotation about z-axis (ANTI-CLOCKWISE)
9. Reflection relative to XY plane
10. Reflection relative to YZ plane
11. Reflection relative to XZ plane
12. Shearing
13. Translation
Enter choice: 1

YOU ARE PERFORMING LOCAL SCALING
Enter the scaling factors:-
fx: 2
fy: 4
fz: 5

TO DISPLAY THE TRANSFORMED FIGURE
Select the type of projection you want to perform.
        1. Orthographic Projection
Enter choice: 1
Scaling with factors 2, 4, 5 in the x, y and z component respectively has been done.
Want to return back to menu? (y/Y - "Yes", any other key - "No") : y


---------------------------
        MAIN MENU
---------------------------
1. Local Scaling
2. Overall Scaling
3. Rotation about x-axis (CLOCKWISE)
4. Rotation about y-axis (CLOCKWISE)
5. Rotation about z-axis (CLOCKWISE)
6. Rotation about x-axis (ANTI-CLOCKWISE)
7. Rotation about y-axis (ANTI-CLOCKWISE)
8. Rotation about z-axis (ANTI-CLOCKWISE)
9. Reflection relative to XY plane
10. Reflection relative to YZ plane
```

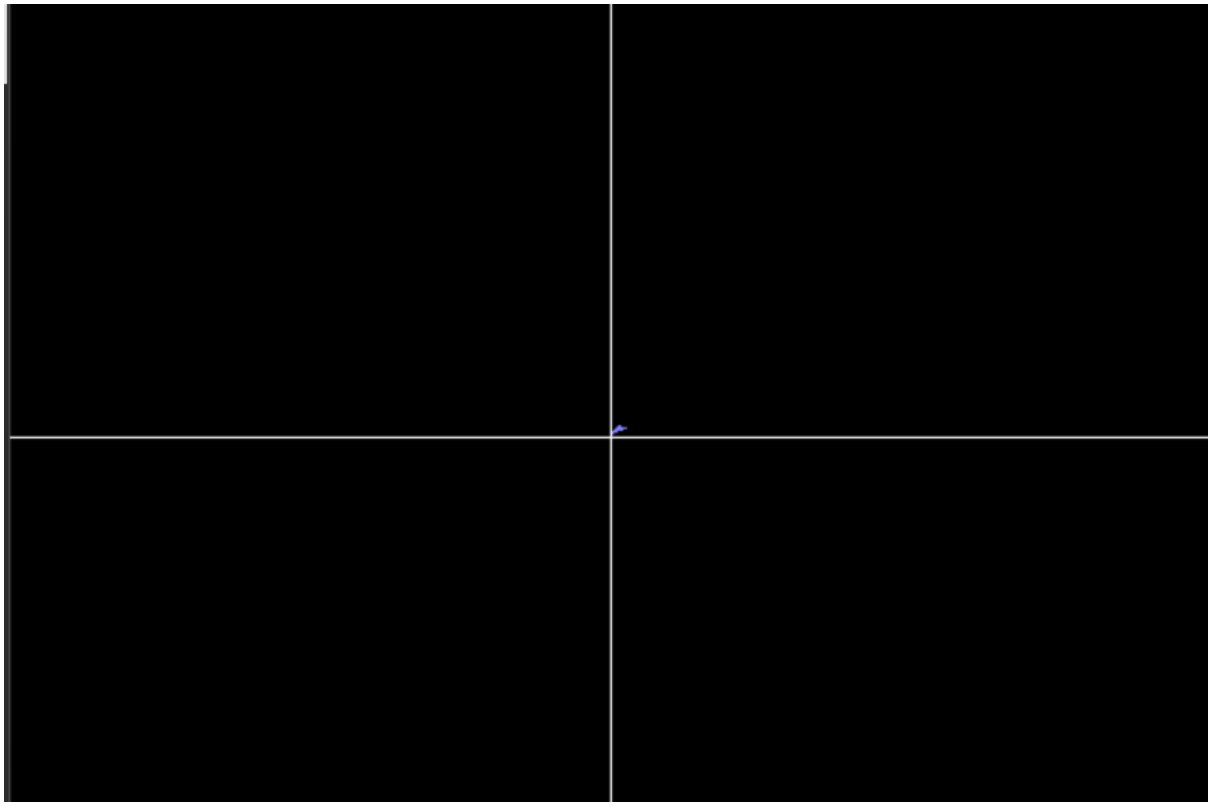
```
9. Reflection relative to XY plane
10. Reflection relative to YZ plane
11. Reflection relative to XZ plane
12. Shearing
13. Translation
Enter choice: 2

YOU ARE PERFORMING OVERALL SCALING
Enter overall scaling factor: 5

TO DISPLAY THE TRANSFORMED FIGURE
Select the type of projection you want to perform.
        1. Orthographic Projection
Enter choice: 1
Overall scaling with a factor 5 has been done.
Want to return back to menu? (y/Y - "Yes", any other key - "No") : y


--------------------------
        MAIN MENU
--------------------------
1. Local Scaling
2. Overall Scaling
3. Rotation about x-axis (CLOCKWISE)
4. Rotation about y-axis (CLOCKWISE)
5. Rotation about z-axis (CLOCKWISE)
6. Rotation about x-axis (ANTI-CLOCKWISE)
7. Rotation about y-axis (ANTI-CLOCKWISE)
8. Rotation about z-axis (ANTI-CLOCKWISE)
9. Reflection relative to XY plane
10. Reflection relative to YZ plane
11. Reflection relative to XZ plane
12. Shearing
13. Translation
Enter choice: 4

YOU ARE PERFORMING ROTATION ABOUT Y-AXIS (CLOCKWISE)
Enter the angle by which you want to rotate the cube: 45

TO DISPLAY THE TRANSFORMED FIGURE
Select the type of projection you want to perform.
        1. Orthographic Projection
Enter choice: 1
Rotation about y-axis in the clockwise direction has been done.
Want to return back to menu? (y/Y - "Yes", any other key - "No") : y
```

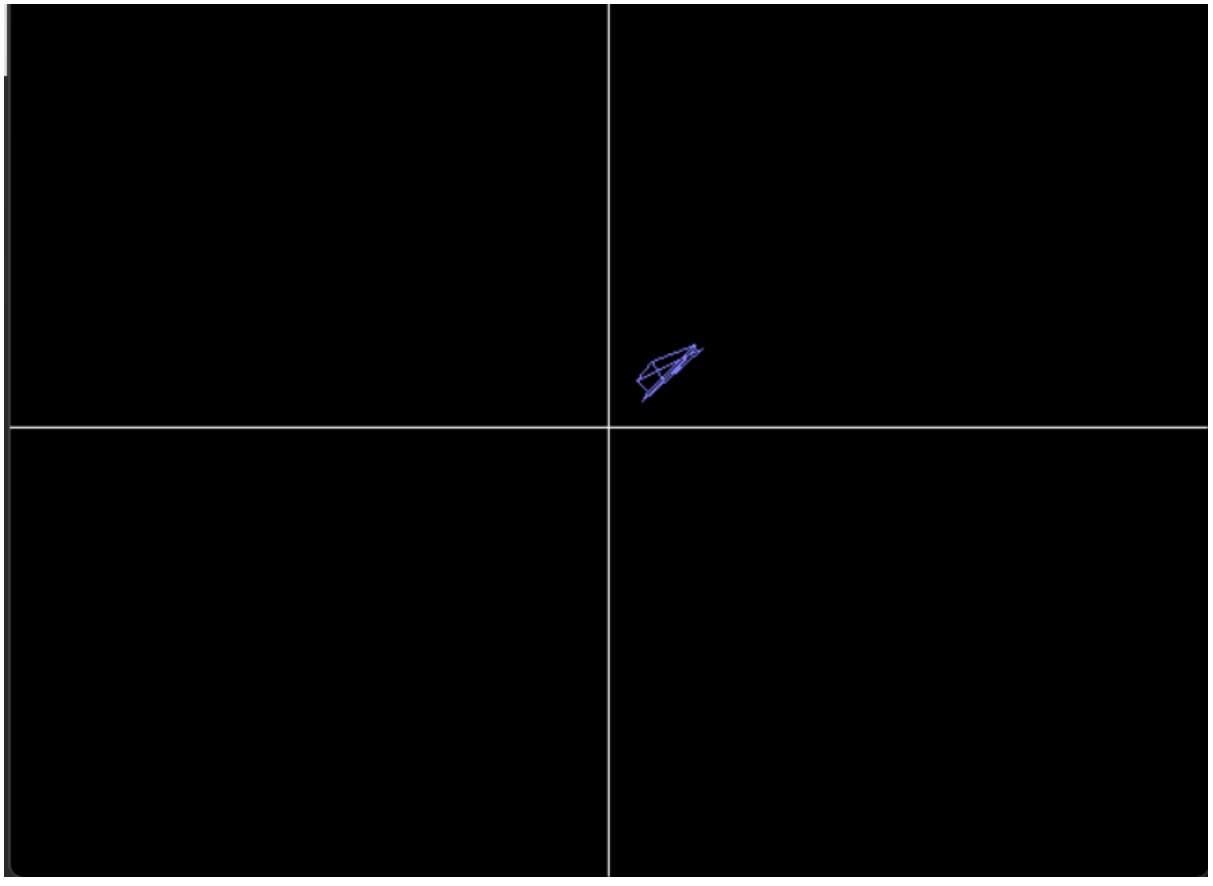
```
---------------------------
       MAIN MENU
-----------------------------
1. Local Scaling
2. Overall Scaling
3. Rotation about x-axis (CLOCKWISE)
4. Rotation about y-axis (CLOCKWISE)
5. Rotation about z-axis (CLOCKWISE)
6. Rotation about x-axis (ANTI-CLOCKWISE)
7. Rotation about y-axis (ANTI-CLOCKWISE)
8. Rotation about z-axis (ANTI-CLOCKWISE)
9. Reflection relative to XY plane
10. Reflection relative to YZ plane
11. Reflection relative to XZ plane
12. Shearing
13. Translation
Enter choice: 9

YOU ARE PERFORMING REFLECTION RELATIVE TO XY PLANE
Reflection relative to the XY plane has been done.
TO DISPLAY THE TRANSFORMED FIGURE
Select the type of projection you want to perform.
       1. Orthographic Projection
Enter choice: 1

Want to return back to menu? (y/Y - "Yes", any other key - "No") : y


---------------------------
       MAIN MENU
-----------------------------
1. Local Scaling
2. Overall Scaling
3. Rotation about x-axis (CLOCKWISE)
4. Rotation about y-axis (CLOCKWISE)
5. Rotation about z-axis (CLOCKWISE)
6. Rotation about x-axis (ANTI-CLOCKWISE)
7. Rotation about y-axis (ANTI-CLOCKWISE)
8. Rotation about z-axis (ANTI-CLOCKWISE)
9. Reflection relative to XY plane
10. Reflection relative to YZ plane
11. Reflection relative to XZ plane
12. Shearing
13. Translation
```

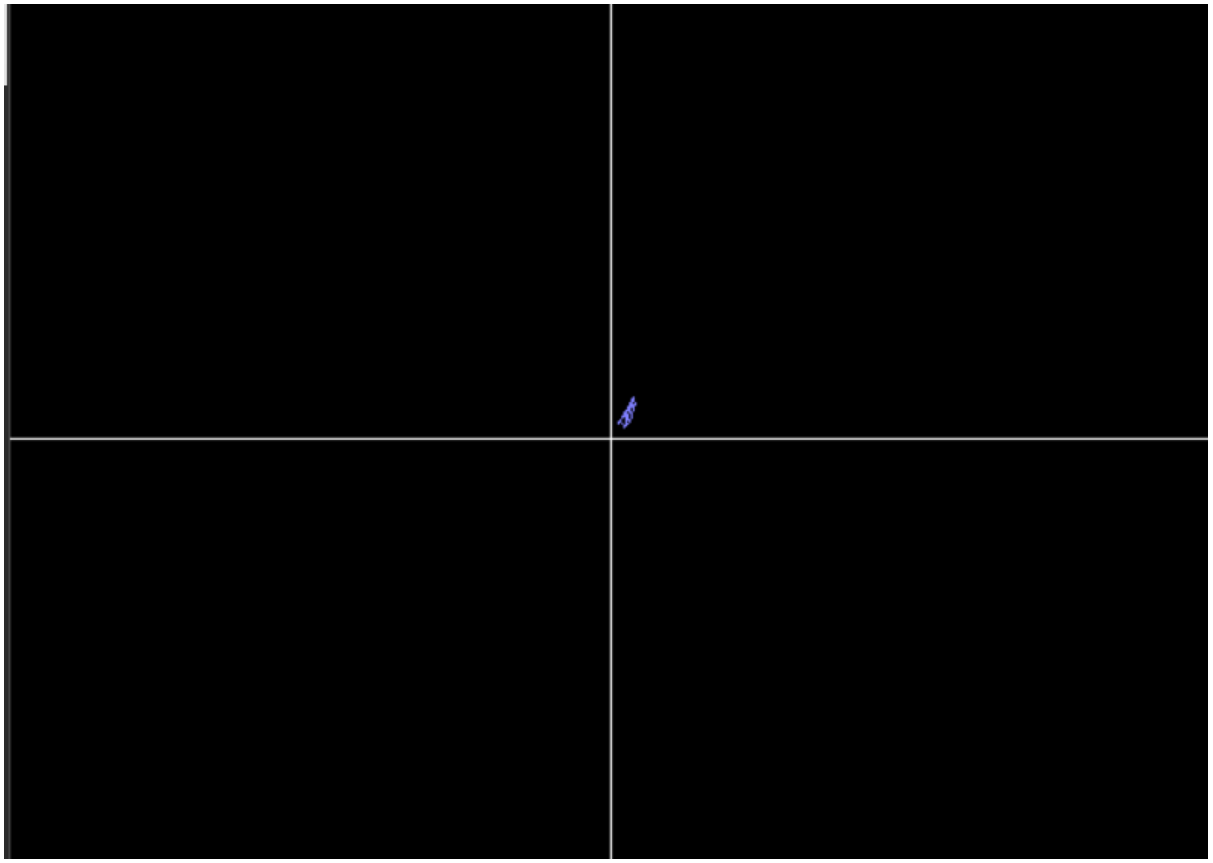
```
TO DISPLAY THE TRANSFORMED FIGURE
Select the type of projection you want to perform.
       1. Orthographic Projection
Enter choice: 1
Translation with factors 1, 2, 3 has been done in the direction of x, y and z-axis respectively.
Want to return back to menu? (y/Y - "Yes", any other key - "No") : y


---------------------------
       MAIN MENU
-----------------------------
1. Local Scaling
2. Overall Scaling
3. Rotation about x-axis (CLOCKWISE)
4. Rotation about y-axis (CLOCKWISE)
5. Rotation about z-axis (CLOCKWISE)
6. Rotation about x-axis (ANTI-CLOCKWISE)
7. Rotation about y-axis (ANTI-CLOCKWISE)
8. Rotation about z-axis (ANTI-CLOCKWISE)
9. Reflection relative to XY plane
10. Reflection relative to YZ plane
11. Reflection relative to XZ plane
12. Shearing
13. Translation
Enter choice: 12

YOU ARE PERFORMING SHEARING
Enter the shearing factors:-
fx1: 1
fx2: 2
fy1: 3
fy2: 4
fz1: 5
fz2: 6

TO DISPLAY THE TRANSFORMED FIGURE
Select the type of projection you want to perform.
       1. Orthographic Projection
Enter choice: 1
Shearing with factors 1, 2, 3, 4, 5, 6 has been done.
Want to return back to menu? (y/Y - "Yes", any other key - "No") : n
```

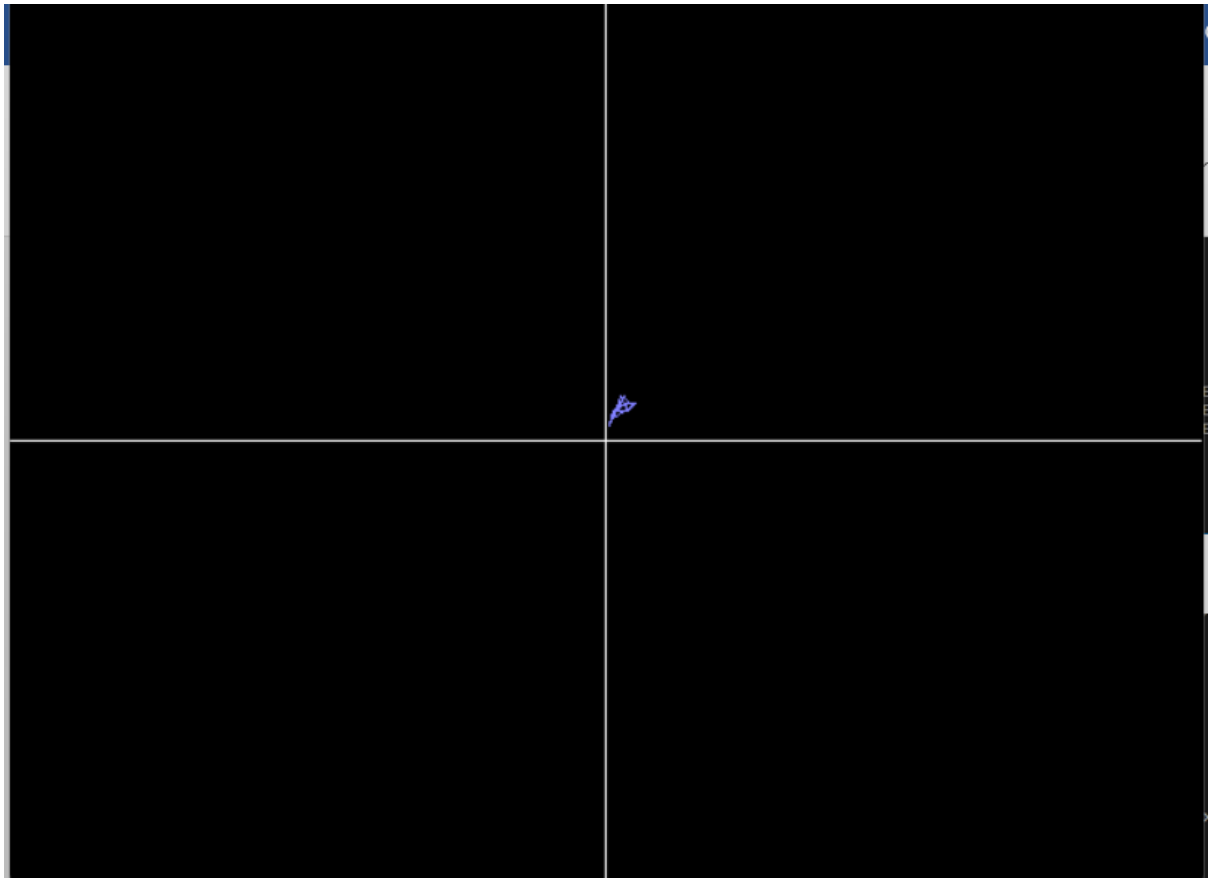












Q8. Write a program to draw Hermite/Bezier curve.

```
#include<iostream>
#include<math.h>
#include<graphics.h>
using namespace std;
const int size = 4;
void bezier(int x[], int y[]) {
  int gr = DETECT, gm;
  initgraph(&gr, &gm, (char*)"C:\\TURBOC3\\BGI");
  double put_x, put_y;
  cout<<"\nCURVE IS BEING DRAWN...";
  for(int i = 0; i < size; i++) {
    putpixel(x[i], y[i], 3);
    delay(1);
  }
  for(double t = 0; t <= 1; t += 0.001) {
    put_x = pow(1 - t, 3) * x[0] + 3 * t * pow(1-t, 2) * x[1] + 3 * pow(t, 2) * (1 - t) * x[2] + pow(t, 3) *
x[3];
    put_y = pow(1 - t, 3) * y[0] + 3 * t * pow(1 - t, 2) * y[1] + 3 * pow(t, 2) * (1 - t) * y[2] + pow(t, 3) *
y[3];
    putpixel(put_x, put_y, WHITE);
    delay(1);
  }
  cout<<"\n\nCURVE IS DRAWN!";
}
void hermite(int x[], int y[]) {
  int gr = DETECT, gm;
  initgraph(&gr, &gm, (char*)("C:\\TURBOC3\\BGI"));
  double put_x, put_y;
  cout<<"\nCURVE IS BEING DRAWN...";
  for(int i = 0; i < size; i++) {
    putpixel(x[i], y[i], 3);
    delay(1);
  }
  for(double t = 0; t <= 1; t += 0.001) {
    put_x = (2 * pow(t, 3) - 3 * pow(t, 2) + 1) * x[0] + (-2 * pow(t, 3) + 3 * pow(t, 2)) * x[1] + (pow(t, 3)
- 2 * pow(t, 2) + t) * x[2] + (pow(t, 3) - pow(t, 2)) * x[3] ;
    put_y = (2 * pow(t, 3) - 3 * pow(t, 2) + 1) * y[0] + (-2 * pow(t, 3) + 3 * pow(t, 2)) * y[1] + (pow(t, 3)
- 2 * pow(t, 2) + t) * y[2] + (pow(t, 3) - pow(t, 2)) * y[3];
    putpixel(put_x, put_y, WHITE);
    delay(1);
  }
 cout<<"\n\nCURVE IS DRAWN!";
}
int main() {
  int x[4], y[4];
```

```
 int choice;
 do {
  cout<<"\nChosse the curve you want to draw:-";
  cout<<"\n1. Bezier Curve";
  cout<<"\n2. Hermite Curve";
  cout<<"\nEnter choice: ";
  cin>>choice;
  switch(choice) {
    case 1 : {
      cout<<"\nEnter the control points:-\n";
      for(int i = 0; i < size; i++) {
        cout<<"x"<<i<<": ";
        cin>>x[i];
        cout<<"y"<<i<<": ";
        cin>>y[i];
      }
      bezier(x, y);
    }
    break;
    case 2 : {
      cout<<"\nEnter the control points:-\n";
      for(int i = 0; i < size; i++) {
        cout<<"x"<<i<<": ";
        cin>>x[i];
        cout<<"y"<<i<<": ";
        cin>>y[i];
      }
      hermite(x, y);
    }
    break;
    default : cout<<"Please enter a valid input!";
  }
  cout<<"\nDo you want to continue? ";
  cin>>choice;
 }while(choice == 'y' || choice == 'Y');
 getch();
 closegraph();
 return 1;
}
```

```
Chosse the curve you want to draw:-
1. Bezier Curve
2. Hermite Curve
Enter choice: 1

Enter the control points:-
x0: 45
y0: 56
x1: 67
y1: 78
x2: 89
y2: 90
```

```
x0: 45
y0: 56
x1: 67
y1: 78
x2: 89
y2: 90
x3: 95
y3: 96

CURVE IS BEING DRAWN...

CURVE IS DRAWN!
```

1.BEIZER CURVE:

2.HERMITE CURVE: