PRACTICAL–6

Aim: write a program to implement composite scaling transformation algorithm.

Software used: Turbo C++

Theory: More complex geometric & coordinate transformations can be built from the basic transformation by using the process of composition of function.

Steps for doing composite scaling transformation:-

1.) Translate the object so that its centre coincides with the origin.

2.) Scale the object with respect to origin.

3.) Translate the scale object back to the original position.

Formula used: T(Xf,Yf).S(Sx,Sy).T(-Xr,-Yr)=S(Xf,Yf)

Source Code:

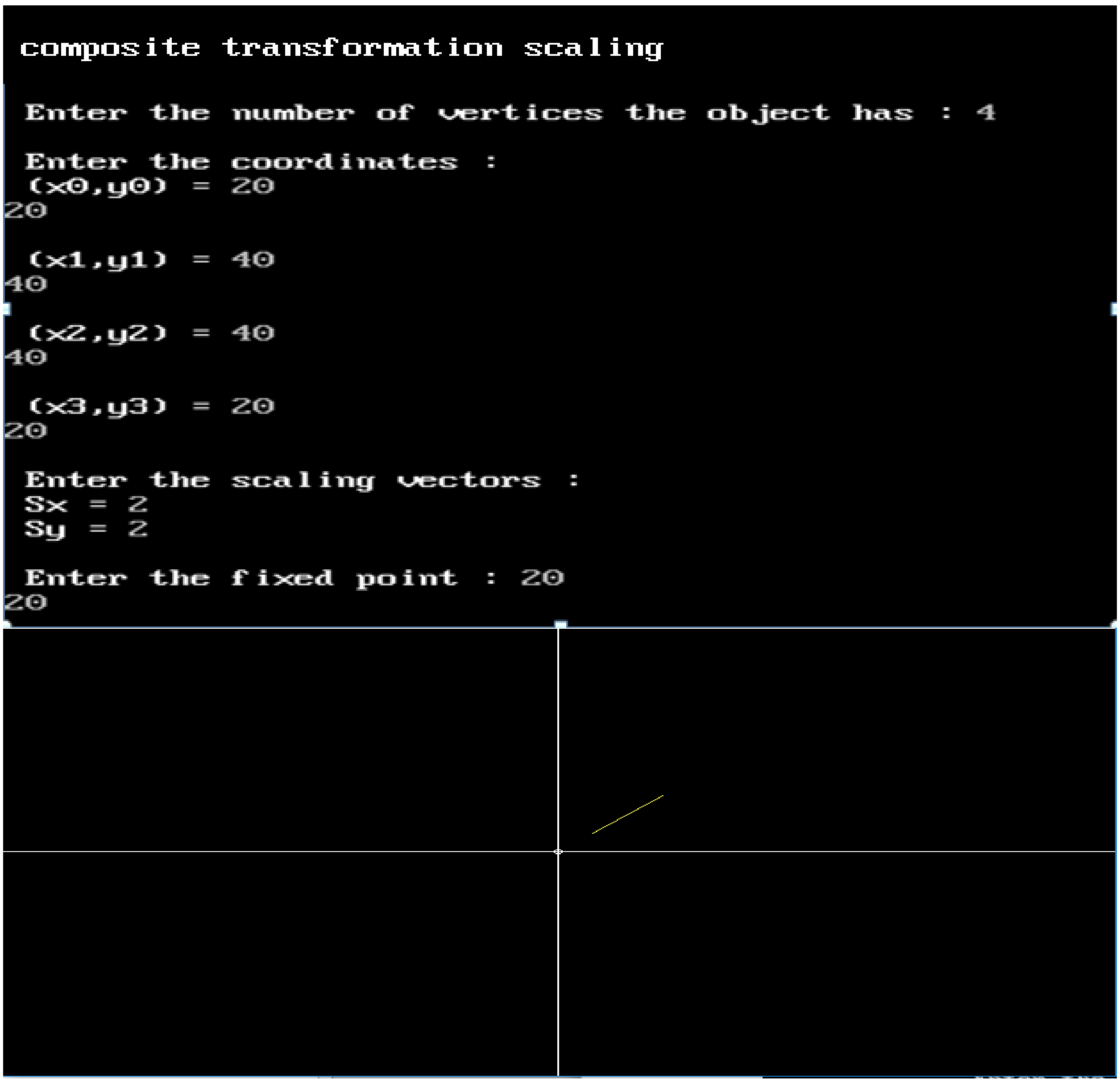
```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
void scaling(int [2][6], int, double, double, int, int, float, float);
void main()
{   clrscr();
  int b[2][6], n;
  double Sx=1, Sy=1;
  int tx, ty;
  int gd = DETECT, gm;
  initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
  cout<<"\n COMPOSITE SCALING TRANSFORMATION \n";
  cout<<"\n Enter the number of vertices the object has : ";
```

```
cin>>n;
cout<<"\n Enter the coordinates : ";
for(int j = 0 ; j < n ; j++)
{       cout<<"\n (x"<<j<<",y"<<j<<") = ";
        cin>>b[0][j]>>b[1][j];       }
cout<<"\n Enter the scaling vectors : ";
cout<<"\n Sx = ";
cin>>Sx;
cout<<" Sy = ";
cin>>Sy;
cout<<"\n Enter the fixed point : ";
cin >> tx >> ty;
float maxx = getmaxx();
float maxy = getmaxy();
cleardevice();
line(maxx/2,0,maxx/2,maxy);
line(0,maxy/2,maxx,maxy/2);
circle(maxx/2, maxy/2, 2);
setcolor(WHITE);
for(int i = 0 ; i < n-1 ; i++)
{
  line(maxx/2+b[0][i], maxy/2-b[1][i], maxx/2+b[0][i+1], maxy/2-b[1][i+1]);
  delay(100);
}
line(maxx/2+b[0][i], maxy/2-b[1][i], maxx/2+b[0][0], maxy/2-b[1][0]);
scaling(b,n,Sx,Sy,tx,ty,maxx,maxy);
getch();
```

```
  closegraph();
}
void scaling(int b[2][6], int n, double Sx, double Sy, int tx, int ty, float maxx, float maxy)
{   double a[3][6];
  double t[3][3];
double c[3][6];
  int i,j;
  for(j = 0 ; j < n ; j++)
  {
    a[0][j] = b[0][j];
    a[1][j] = b[1][j];
  }
  for(i = 0 ; i < n ; i++)
    a[2][i] = 1;
  for( i = 0 ; i < 3 ; i++)
    for( j = 0 ; j < 3 ; j++)
      t[i][j]=0;
  t[0][0] = Sx;
  t[1][1] = Sy;
  t[0][2] = tx * (1-Sx);
  t[1][2] = ty * (1-Sy);
  t[2][2] = 1;
  for(i = 0 ; i < 3 ; i++)
    for(j = 0 ; j < n ; j++)
    {
      c[i][j] = 0;
      for(int k = 0; k < 3 ; k++)
```

```
            c[i][j] += t[i][k] * a[k][j];
    }
  setcolor(YELLOW);
  for(i = 0 ; i < n-1 ; i++)
  {    line(maxx/2+c[0][i], maxy/2-c[1][i], maxx/2+c[0][i+1], maxy/2-c[1][i+1]);
    delay(100);   }
  line(maxx/2+c[0][i], maxy/2-c[1][i], maxx/2+c[0][0], maxy/2-c[1][0]);    }
```

Output:



PRACTICAL–8

Aim: write a program to implement composite rotation transformation algorithm.

Software used: Turbo C++

Theory: More complex geometric & coordinate transformations can be built from the basic transformation by using the process of composition of function.

Steps for doing composite rotation transformation:-

1.) Translate the object so that pivot point position is moved to the co-ordinate origin.
2.) Rotate the object about the co-ordinate origin.
3.) Translate the object so that the pivot point is returned to the original position.

Formula used: T(Xf,Yf).R θ.T(-Xr,-Yr)=S(Xf,Yf)

Source Code:

```
#include<iostream.h>

#include<graphics.h>

#include<conio.h>

#include<dos.h>

#include<math.h>

#define pi 3.14285714

void rotation(int [2][6], int, double, int, int, float, float);

void main()

{   clrscr();

  int b[2][6], n;

  double angle;

  int tx, ty;

  int gd = DETECT, gm;

  initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

 cout<<"\n COMPOSITE ROTATION TRANSFORMATION \n";
```

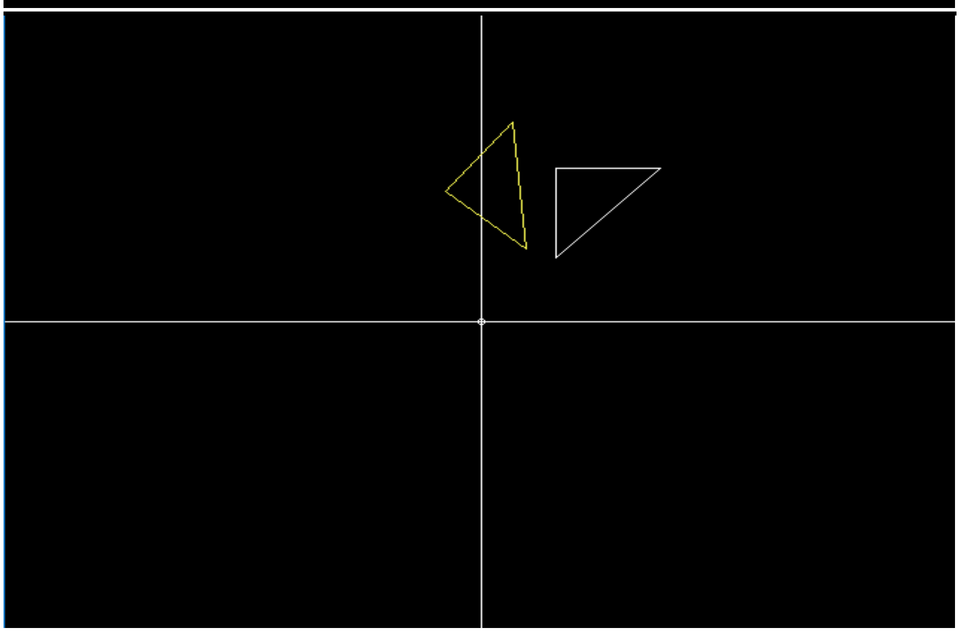
```
 cout<<"\n Enter the number of vertices the object has : ";
  cin>>n;
  cout<<"\n Enter the coordinates : ";
   for(int j = 0 ; j < n ; j++)
{      cout<<"\n (x"<<j<<",y"<<j<<") = ";
       cin>>b[0][j]>>b[1][j];     }
cout<<"\n Enter the rotation angle : ";
  cin>>angle;
  cout<<"\n Enter the fixed point : ";
  cin >> tx >> ty;
  float maxx = getmaxx();
  float maxy = getmaxy();
  cleardevice();
  line(maxx/2,0,maxx/2,maxy);
  line(0,maxy/2,maxx,maxy/2);
  circle(maxx/2, maxy/2, 2);

  setcolor(WHITE);
  for(int i = 0 ; i < n-1 ; i++)
  {
    line(maxx/2+b[0][i], maxy/2-b[1][i], maxx/2+b[0][i+1], maxy/2-b[1][i+1]);
    delay(100);
  }
  line(maxx/2+b[0][i], maxy/2-b[1][i], maxx/2+b[0][0], maxy/2-b[1][0]);
  rotation(b,n,angle,tx,ty,maxx,maxy);
  getch();  }
void rotation(int b[2][6], int n, double angle, int tx, int ty, float maxx, float maxy)
```

```
{
  double a[3][6];
  double t[3][3];
  double c[3][6];
int i,j;
 angle = ((pi/180) * angle);
  for(j = 0 ; j < n ; j++)
    {
    a[0][j] = b[0][j];
    a[1][j] = b[1][j];
  }
  for(i = 0 ; i < n ; i++)
    a[2][i] = 1;
  for( i = 0 ; i < 3 ; i++)
    for( j = 0 ; j < 3 ; j++)
      t[i][j]=0;
  t[0][0] = t[1][1] = cos(angle);
  t[1][0] = sin(angle);
  t[0][1] = (-1 * sin(angle));
  t[0][2] = tx * (1-cos(angle)) + ty * (sin(angle));
  t[1][2] = ty * (1-cos(angle)) - tx * (sin(angle));
  t[2][2] = 1;
  for(i = 0 ; i < 3 ; i++)
    for(j = 0 ; j < n ; j++)
    {
      c[i][j] = 0;
      for(int k = 0; k < 3 ; k++)
```

```
          c[i][j] += t[i][k] * a[k][j];
  }
 setcolor(YELLOW);
 for(i = 0 ; i < n-1 ; i++)
 {
   line(maxx/2+c[0][i], maxy/2-c[1][i], maxx/2+c[0][i+1], maxy/2-c[1][i+1]);
   delay(100);
 }
 line(maxx/2+c[0][i], maxy/2-c[1][i], maxx/2+c[0][0], maxy/2-c[1][0]);    }
```

Output:

```
Composite rotation transformation

Enter the number of vertices the object has : 3

Enter the coordinates :
(x0,y0) = 50
50

(x1,y1) = 50
120

(x2,y2) = 120
120

Enter the rotation angle : 50

Enter the fixed point : 32
32
```



PRACTICAL–9

Aim: write a program to implement Cohen Sutherland Line Clipping algorithm.

Software used: Turbo C++

Theory: This is one of the oldest and most popular line clipping algorithm. To speed up the process this algorithm performs initial tests that reduce number of intersections that must be calculated. It does so by using a 4 bit code called as region code or outcodes. These codes identify location of the end point of line. Each bit position indicates a direction, starting from the rightmost position of each bit indicates left, right, bottom, top respectively. Once we establish region codes for both the endpoints of a line we determine whether the endpoint is visible, partially visible or invisible with the help of logical AND operation of the region codes.

Source Code:

```
#include<iostream.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);

void main()
{
   int gd=DETECT,v,gm;
   PT p1,p2,p3,p4,ptemp;

   cout<<"\nEnter x1 and y1\n";
   cin>>p1.x>>p1.y;
   cout<<"\nEnter x2 and y2\n";
   cin>>p2.x>>p2.y;

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

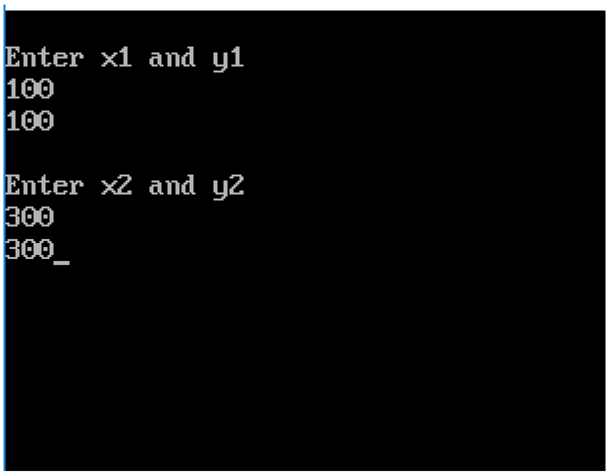
```
    delay(500);

    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);
}
```

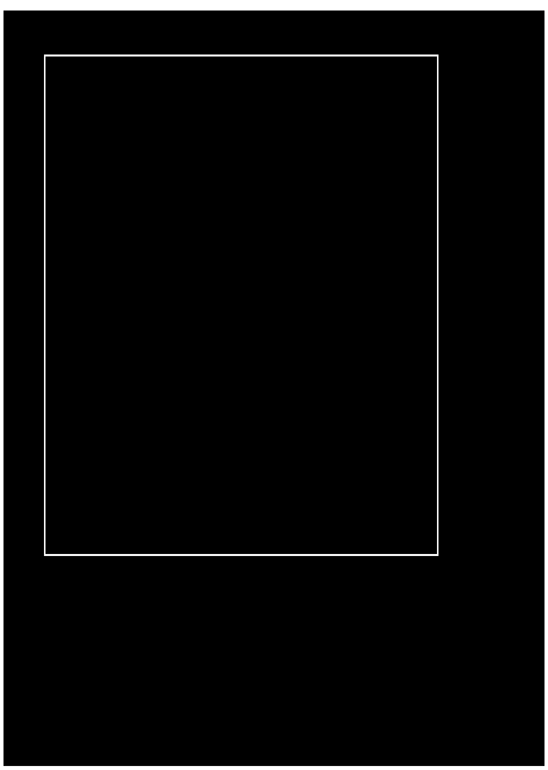
```
PT setcode(PT p)    //for setting the 4 bit code
{
   PT ptemp;

   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'))
```

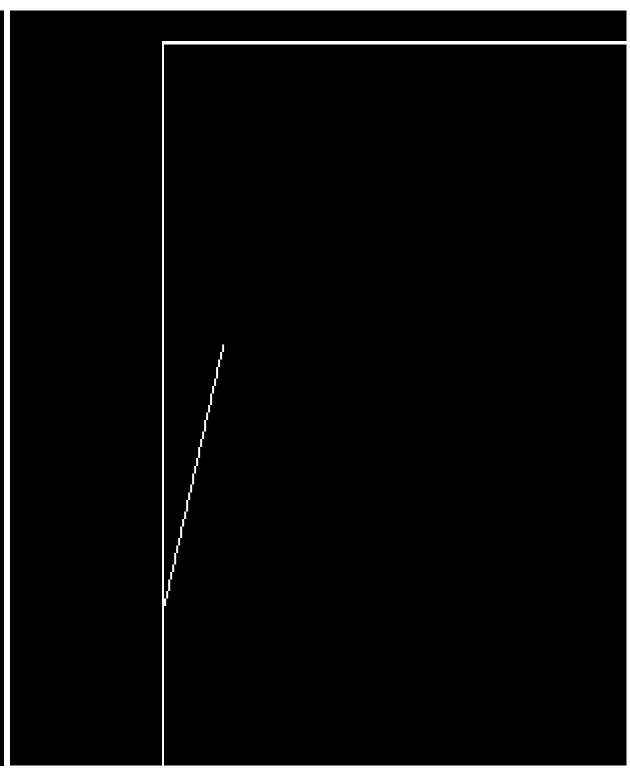
```
        flag='0';
    }

    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);
}
```

<u>Output:</u>







PRACTICAL–10

Aim: write a program to implement Sutherland Hodgman polygon Clipping algorithm.

Software used: Turbo C++

Theory: Sutherland and Hodgman's polygon-clipping algorithm uses a divide-and-conquer strategy: It solves a series of simple and identical problems that, when combined, solve the overall problem. The simple problem is to clip a polygon against a single infinite clip edge. Four clip edges, each defining one boundary of the clip rectangle, successively clip a polygon against a clip rectangle.

Source Code:

```
#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;
   }
```

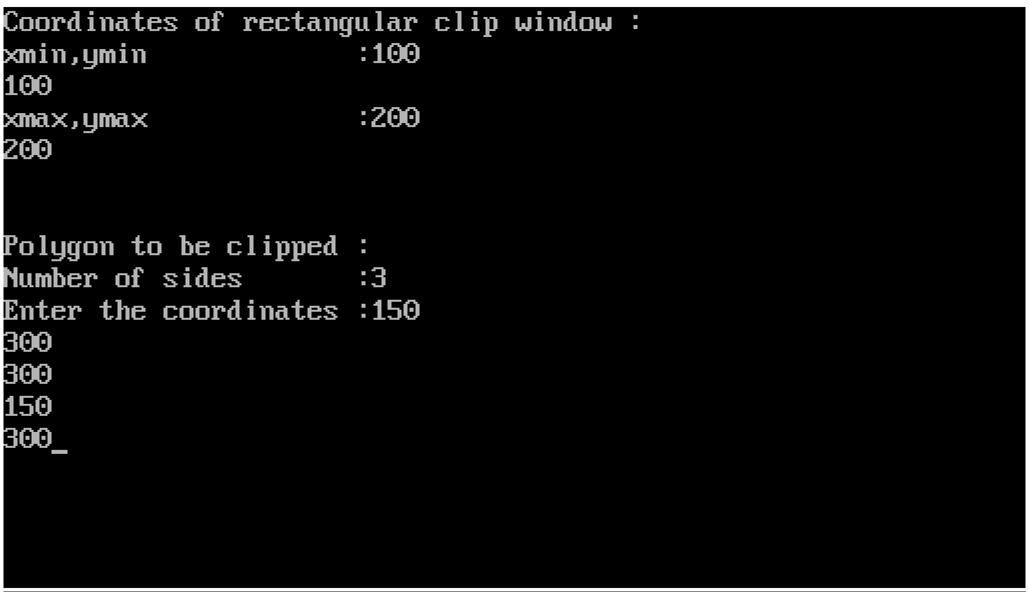
```
}

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);
```

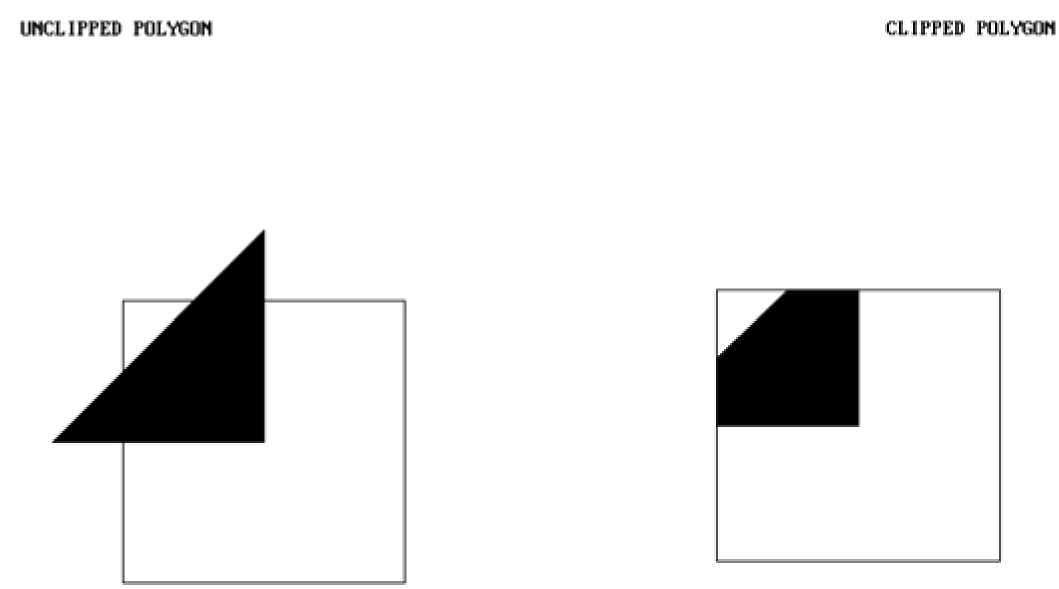
```
        arr[k+2]=x2;
        arr[k+3]=y2;
        k+=4;
    }
    if(x1 <= xmax  && x2 > xmax)
    {
        arr[k]=xmax;
        arr[k+1]=y1+m*(xmax-x1);
        k+=2;
    }
}

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(WHITE);
    rectangle(xmin,ymax,xmax,ymin);
    cout<<"\tCLIPPED POLYGON";
    getch();
    closegraph();
}
```

Output:





PRACTICAL–7

Aim: write a program to implement basic reflection transformation algorithm.

Software used: Turbo C++

Theory: Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.

Source Code:

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int a,a1,b,b1,c,c1,xt,ch;
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
a=getmaxx();
a1=getmaxy();
b=a/2;
b1=a1/2;

line(b,0,b,a1);
line(0,b1,a,b1);
line(400,200,600,200);
line(400,200,400,100);
line(400,100,600,200);

cout<<"1.origin\n";
cout<<"2.x-axis\n";
cout<<"3.y-axis\n";
cout<<"4.exit\n";

do
{
cout<<"Enter your choice\n";
cin>>ch;
switch(ch)
{
case 1:
    c=400-b;c1=200-b1;
    line(b-c,b1-c1,b-c-200,b1-c1);
    line(b-c,b1-c1,b-c,b1-c1+100);
    line(b-c,b1-c1+100,b-c-200,b1-c1);
```

```
	break;
case 2:
	c=400-b;c1=200-b1;
	line(b+c,b1-c1,b+c+200,b1-c1);
	line(b+c,b1-c1,b+c,b1-c1+100);
	line(b+c,b1-c1+100,b+c+200,b1-c1);
	break;


case 3:
	c=400-b;c1=200-b1;
	line(b-c,b1+c1,b-c-200,b1+c1);
	line(b-c,b1+c1,b-c,b1+c1-100);
	line(b-c,b1+c1-100,b-c-200,b1+c1);
	break;
}

}while(ch<4);
getch();
closegraph();
}
```

Output: