INDEX

Programming Environment:

OS: **Linux Mint 17.3**
Compiler: **clang++ -std=c++14**

Editor: Atom Text Editor
Programming Language: C++

9.	**Write a program to implement composite transformations**

Description:

Transformations are used to transform one set of coordinates into the other. The various types of transformations are translation, rotation, scaling, shear, reflection etc. Using homogeneous transformation matrices T_i we can transform P into P' as follows.

$$P' = (T_1 T_2 T_3 T_4 \ldots . T_n) P$$

Where P is the vector containing original coordinates and P' is the vector containing transformed set of coordinates

Program:

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
const double pi = 3.1415;

vector< vector<double> > mul(vector< vector<double> >& a, vector< vector<double> >& b){
   vector< vector<double> > ans;
   int n  = a[0].size();
   int p = b.size();
   int q = b[0].size();
   assert(n==p);
   for(int i = 0; i<n; i++)
   {
      vector<double> r;
      for(int j =0; j<q; j++)
      {
         double num = 0;
         for(int k = 0; k<p; k++)
         num += a[i][k]*b[k][j];
         r.push_back(num);
      }
      ans.push_back(r);
   }
   return ans;
}
vector< vector<double> > make_P(int x, int y)
{
   vector<double> p,q,r;
```

```
    vector< vector<double> > ans;
    p.push_back(x);
    q.push_back(y);
    r.push_back(1);
    ans.push_back(p);
    ans.push_back(q);
    ans.push_back(r);
    return ans;
}
vector< vector<double> > make_T(){
    vector< vector<double> > T(3);
    T[0].resize(3);
    T[1].resize(3);
    T[2].resize(3);
    for(int i = 0; i<3; i++)
        for(int j = 0; j<3; j++)
        T[i][j] = 0;
    T[0][0] = 1;
    T[1][1] = 1;
    T[2][2] = 1;
    return T;
}
vector< vector<double> > rotateComposite(int x, int y, double xr, double yr, double theta){
    vector< vector<double> > P = make_P(x,y);
    vector< vector<double> > T = make_T();
    double c = cos(theta*pi/180.0);
    double s = sin(theta*pi/180.0);
    T[0][0] = c;
    T[0][1] = -s;
    T[0][2] = xr*(1-c) + yr*s;
    T[1][0] = s;
    T[1][1] = c;
    T[1][2] = yr*(1-c)  - xr*s;
    return mul(T,P);
}

vector< vector<double> > scaleComposite(int x, int y, double xr, double yr,double sx, double sy){
    vector< vector<double> > P = make_P(x,y);
    vector< vector<double> > T = make_T();
    T[0][0] = sx;
    T[0][2] = xr*(1-sx);
    T[1][1] = sy;
```

```
    T[1][2] = yr*(1-sy);
    return mul(T,P);
}
vector< vector<double> > shear(int x, int y, double xr, double yr,double hx,double hy){
    vector< vector<double> > P = make_P(x,y);
    vector< vector<double> > T = make_T();
    T[0][0] = 1;
    T[0][1] = hx;
    T[0][2] = -hx*yr;
    T[1][0] = hy;
    T[1][1] = 1;
    T[1][2] = -hy*xr;
    return mul(T,P);
}

vector< vector<double> > reflectX(int x, int y, double yr){
    vector< vector<double> > P = make_P(x,y);
    vector< vector<double> > T = make_T();
    T[0][0] = 1;
    T[1][1] = -1;
    T[1][2] = 2*yr;
    return mul(T,P);
}
vector< vector<double> > reflectY(int x, int y, double xr){
    vector< vector<double> > P = make_P(x,y);
    vector< vector<double> > T = make_T();
    T[0][0] = -1;
    T[0][2] = 2*xr;
    return mul(T,P);
}

int main(){
    int arr[3][3]=  {{1,2,3},{4,5,6},{7,8,9}};
    int g = DETECT;
    int d = 0;
    initgraph(&g,&d," ");

    int option;
    while(1){
        cout << "1 for composite rotation\n2 for scaling\n3 for shear\n4 for x reflection\n5for y
                    reflection";
        cin >> option;
```

```
if(option==1){
    cout << "Enter xmin,ymin,xmax,ymax for the rectangle";
    int x,y,x2,y2,x3,y3,x4,y4;
    cin >> x >> y>> x3 >> y3;
    x2 = x3;
    y2 = y;
    x4 = x;
    y4 = y3;

    cout << "Enter xr and yr and theta\n";
    double xr,yr,theta;
    cin >> xr >> yr >> theta;

    vector< vector<double> > P_dash = rotateComposite(x,y,xr,yr,theta);
    vector< vector<double> > P_dash2 = rotateComposite(x2,y2,xr,yr,theta);
    vector< vector<double> > P_dash3 = rotateComposite(x3,y3,xr,yr,theta);
    vector< vector<double> > P_dash4 = rotateComposite(x4,y4,xr,yr,theta);

    cout << "New points are: " << P_dash[0][0] << "," << P_dash[1][0] << "\n";
    cout << P_dash2[0][0] << "," << P_dash2[1][0]<< "\n";
    cout << P_dash3[0][0] << "," << P_dash3[1][0] << "\n";
    cout << P_dash4[0][0] << "," << P_dash4[1][0] << "\n";
    int arr[10] = {P_dash[0][0],P_dash[1][0],P_dash2[0][0],P_dash2[1][0],P_dash3[0]
            [0],P_dash3[1][0],P_dash4[0][0],P_dash4[1][0],P_dash[0][0],P_dash[1][0]};
    rectangle(x,y,x3,y3);
    drawpoly(5,arr);
    delay(1000);
}
else if(option==2){
    cout << "Enter xmin,ymin,xmax,ymax for the rectangle";
    int x,y,x2,y2,x3,y3,x4,y4;
    cin >> x >> y>> x3 >> y3;
    x2 = x3;
    y2 = y;
    x4 = x;
    y4 = y3;
    cout << "Enter xr and yr and sx and sy\n";
    double xr,yr,sx,sy;
    cin >> xr >> yr >> sx >> sy;
    vector< vector<double> > P_dash = scaleComposite(x,y,xr,yr,sx,sy);
    vector< vector<double> > P_dash2 = scaleComposite(x2,y2,xr,yr,sx,sy);
    vector< vector<double> > P_dash3 = scaleComposite(x3,y3,xr,yr,sx,sy);
```

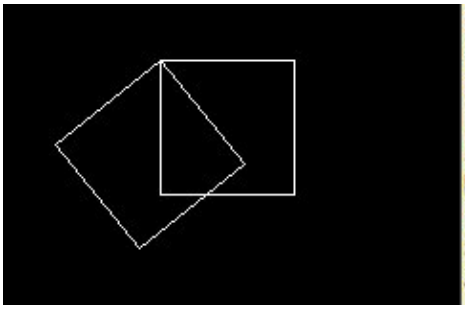
```
        vector< vector<double> > P_dash4 = scaleComposite(x4,y4,xr,yr,sx,sy);
        cout << "New points are: " << P_dash[0][0] << "," << P_dash[1][0] << "\n";
        cout << P_dash2[0][0] << "," << P_dash2[1][0]<< "\n";
        cout << P_dash3[0][0] << "," << P_dash3[1][0] << "\n";
        cout << P_dash4[0][0] << "," << P_dash4[1][0] << "\n";
        int arr[10] = {P_dash[0][0],P_dash[1][0],P_dash2[0][0],P_dash2[1][0],P_dash3[0]
[0],P_dash3[1][0],P_dash4[0][0],P_dash4[1][0],P_dash[0][0],P_dash[1][0]};
        rectangle(x,y,x3,y3);
        drawpoly(5,arr);
        delay(1000);
      }
      else if(option==3){
        cout << "Enter xmin,ymin,xmax,ymax for the rectangle";
        int x,y,x2,y2,x3,y3,x4,y4;
        cin >> x >> y>> x3 >> y3;
        x2 = x3;
        y2 = y;
        x4 = x;
        y4 = y3;
        cout << "Enter xr and yr and shx and shy\n";
        double xr,yr,sx,sy;
        cin >> xr >> yr >> sx >> sy;

        vector< vector<double> > P_dash =  shear(x,y,xr,yr,sx,sy);
        vector< vector<double> > P_dash2 = shear(x2,y2,xr,yr,sx,sy);
        vector< vector<double> > P_dash3 = shear(x3,y3,xr,yr,sx,sy);
        vector< vector<double> > P_dash4 = shear(x4,y4,xr,yr,sx,sy);
        cout << "New points are: " << P_dash[0][0] << "," << P_dash[1][0] << "\n";

        cout << P_dash2[0][0] << "," << P_dash2[1][0]<< "\n";
        cout << P_dash3[0][0] << "," << P_dash3[1][0] << "\n";
        cout << P_dash4[0][0] << "," << P_dash4[1][0] << "\n";
        int arr[10] = {P_dash[0][0],P_dash[1][0],P_dash2[0][0],P_dash2[1][0],P_dash3[0]
[0],P_dash3[1][0],P_dash4[0][0],P_dash4[1][0],P_dash[0][0],P_dash[1][0]};
        rectangle(x,y,x3,y3);
        drawpoly(5,arr);
        delay(1000);
      }
      else if(option==4)
      {
        cout << "Enter xmin,ymin,xmax,ymax for the rectangle";
        int x,y,x2,y2,x3,y3,x4,y4;
```

```
        cin >> x >> y>> x3 >> y3;
        x2 = x3;
        y2 = y;
        x4 = x;
        y4 = y3;
        double yr;
        cout << "Enter yr for reflection\n";
        cin >> yr;
        vector< vector<double> > P_dash  =  reflectX(x,y,yr);
        vector< vector<double> > P_dash2 = reflectX(x2,y2,yr);
        vector< vector<double> > P_dash3 = reflectX(x3,y3,yr);
        vector< vector<double> > P_dash4 = reflectX(x4,y4,yr);
        cout << "New points are: " << P_dash[0][0] << "," << P_dash[1][0] << "\n";
        cout << P_dash2[0][0] << "," << P_dash2[1][0]<< "\n";
        cout << P_dash3[0][0] << "," << P_dash3[1][0] << "\n";
        cout << P_dash4[0][0] << "," << P_dash4[1][0] << "\n";
        int arr[10] = {P_dash[0][0],P_dash[1][0],P_dash2[0][0],P_dash2[1][0],P_dash3[0]
[0],P_dash3[1][0],P_dash4[0][0],P_dash4[1][0],P_dash[0][0],P_dash[1][0]};
        rectangle(x,y,x3,y3);
        drawpoly(5,arr);
        delay(1000);
      }
      else if(option==5)
      {
        cout << "Enter xmin,ymin,xmax,ymax for the rectangle";
        cin >> x >> y>> x3 >> y3;
        double xr;
        cout << "Enter xr for reflection\n";
        cin >> xr;
        vector< vector<double> > P_dash  =  reflectY(x,y,xr);
        vector< vector<double> > P_dash2 = reflectY(x2,y2,xr);
        vector< vector<double> > P_dash3 = reflectY(x3,y3,xr);
        vector< vector<double> > P_dash4 = reflectY(x4,y4,xr);

        cout << "New points are: " << P_dash[0][0] << "," << P_dash[1][0] << "\n";
        cout << P_dash2[0][0] << "," << P_dash2[1][0]<< "\n";
        cout << P_dash3[0][0] << "," << P_dash3[1][0] << "\n";
        cout << P_dash4[0][0] << "," << P_dash4[1][0] << "\n";

         int arr[10] = {P_dash[0][0],P_dash[1][0],P_dash2[0][0],P_dash2[1][0],P_dash3[0]
        [0],P_dash3[1][0],P_dash4[0][0],P_dash4[1][0],P_dash[0][0],P_dash[1][0]};
        rectangle(x,y,x3,y3);
```

```
            drawpoly(5,arr);
            delay(1000);
        }
    }
}
```

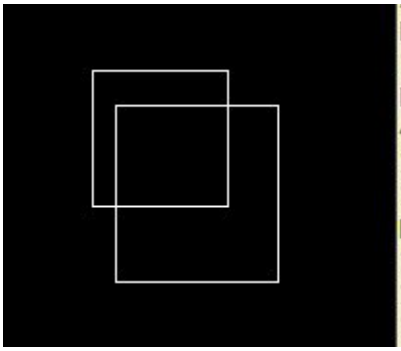
OUTPUT:

A: Composite Rotation about a point:



```
Enter xmin,ymin,xmax,ymax for the rectangle
100 100 170 170
Enter xr and yr and theta
100 100
51
New points are: 100,100
144.054,154.399
89.6548,198.453
45.6009,144.054
```

A: Composite scaling about a point:



```
Enter xmin,ymin,xmax,ymax for the rectangle
100 100 170 170
Enter xr and yr and sx and sy
40 40
1.2
1.3
New points are: 112,118
196,118
196,209
112,209
```

10. Write a program to clip a line using Cohen Sutherland Algorithm

Description:

The Cohen–Sutherland algorithm is a computer graphics algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions (or a three-dimensional space into 27 regions), and then efficiently determines the lines and portions of lines that are visible in the center region of interest (the viewport).

The algorithm includes, excludes or partially includes the line based on where:

•Both endpoints are in the viewport region (bitwise OR of endpoints == 0): trivial accept.

•Both endpoints share at least one non-visible region which implies that the line does not cross the visible region. (bitwise AND of endpoints != 0): trivial reject.

•Both endpoints are in different regions: In case of this nontrivial situation the algorithm finds one of the two points that is outside the viewport region (there will be at least one point outside). The intersection of the outpoint and extended viewport border is then calculated (i.e. with the parametric equation for the line) and this new point replaces the outpoint. The algorithm repeats until a trivial accept or reject occurs.

Program:

```
#include <bits/stdc++.h>
#include <graphics.h>
#define bitCode int
using namespace std;

/* Algorithm for line clipping using cohen sutherland algorithm.Clip a line l given boundary b*/

//Set Positions to mark a point as inside or outside
#define INSIDE 0    //0000
#define LEFT 1      //0001
#define RIGHT 2     //0010
#define BOTTOM 4    //0100
#define TOP 8       //1000

typedef struct Boundary{
   int x_min, x_max, y_min, y_max;
}Boundary;
```

```
typedef struct Point{
   int x, y;
   Point (){}
   Point(int a, int b) : x(a), y(b){}
}Point;

typedef struct Line{
   Point A, B;
   string lineStatus(){
      return "(" + to_string(A.x)  + "," + to_string(A.y) + ") to (" + to_string(B.x) + "," +
         to_string(B.y) + ")\n";
   }
}Line;


int getBitCode(Point p, Boundary &b){
   //Compute bitcodes for the given point depending on where the point lies wrt boundary
   int curBit = INSIDE;

   if (p.x < b.x_min)
      curBit |= LEFT;
   if (p.x > b.x_max)
      curBit |= RIGHT;
   if (p.y < b.y_min)
      curBit |= BOTTOM;
   if (p.y > b.y_max)
      curBit |= TOP;

   return curBit;
}

void showClipping_Line(Line &l, Boundary &b, string &message){
   rectangle(b.x_min, b.y_max, b.x_max, b.x_min);
   line(l.A.x, l.A.y, l.B.x, l.B.y);
   printf("\t %s ", message.c_str());
   sleep(2);
   cleardevice();
}

void cohenSuther_clipping(Boundary &b, Line &l){
```

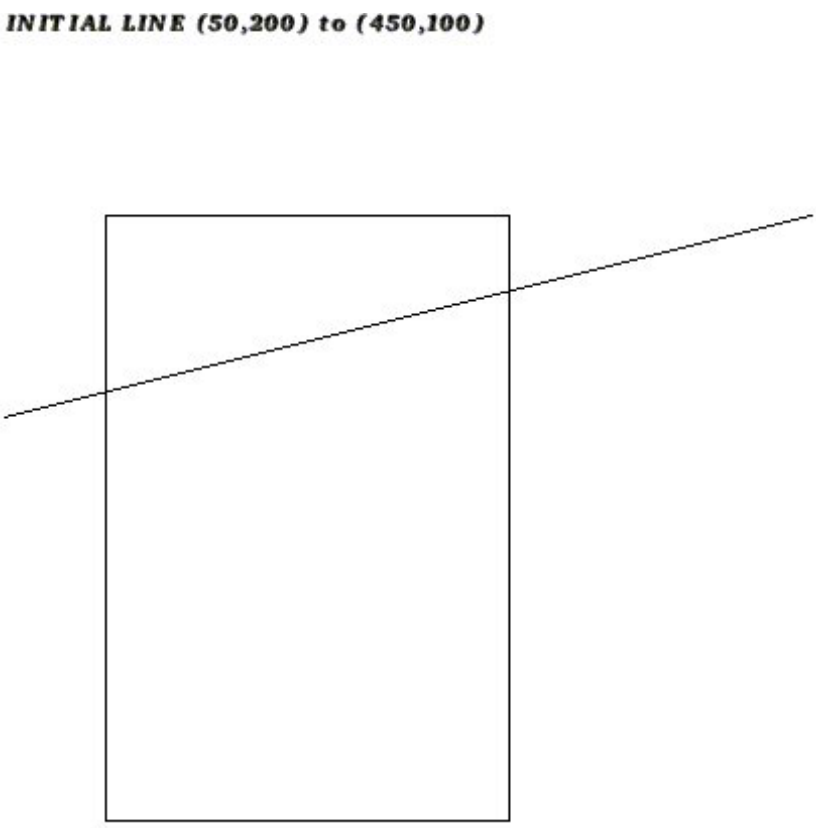
```
//Compute bitcodes of two points of the given line and take decisions accordigly
cout << "\n Initial line : " << l;

string message = " INITIAL LINE " + l.lineStatus();
showClipping_Line(l, b, message);

bitCode bitA = getBitCode(l.A, b);
bitCode bitB = getBitCode(l.B, b);
cout << "\n BitA: " << bitA;
cout << "\n BitB: " << bitB;

bool accept = false;
while (true){
   cout << "\n\n Current Line : " << l;
   if (!(bitA | bitB)){ //line is completely inside
      message = " Line is completely inside ";
      accept = true;
      break;
   }

   if (bitA & bitB){   //line lies outside the boundary
      message = " Line is completely outside ";
      accept = false;
      break;
   }

   //Now line has some intersection with the boundary. Either one or both points
   //maybe outside. Clip the line from outside to the intersection
   message = " Some part of line is inside ";

   Point intersection;
   double slope = (l.B.y - l.A.y * 1.0)/( l.B.x - l.A.x);
   bitCode anyOuterPoint = bitA ? bitA : bitB; //Any point which is outside

   if (anyOuterPoint & TOP){ //then point is above boundary
      intersection.y = b.y_max;
      intersection.x = l.A.x + (intersection.y - l.A.y)/slope;
   }

   else if (anyOuterPoint & BOTTOM){ //point is below boundary
      intersection.y = b.y_min;
```

```
            intersection.x = l.A.x + (intersection.y - l.A.y)/slope;
        }

        else if (anyOuterPoint & LEFT){ //point is left to the boundary
            intersection.x = b.x_min;
            intersection.y = l.A.y + slope*(intersection.x - l.A.x);
        }

        else if (anyOuterPoint & RIGHT){ //point is right to the boundary
            intersection.x = b.x_max;
            intersection.y = l.A.y + slope*(intersection.x - l.A.x);
        }

        // Now we move outside point to intersection point to clip
                    // and get ready for next pass.
        if (anyOuterPoint == bitA){
            l.A.x = intersection.x;
            l.A.y = intersection.y;
            bitA = getBitCode(l.A, b);
        }
        else{
            l.B.x = intersection.x;
            l.B.y = intersection.y;
            bitB = getBitCode(l.B, b);
        }
        cout << " New line: " << l;
        message.append(l.lineStatus());
        showClipping_Line(l, b, message);
    }

    if (accept){
        cout << "\n The line is accepted";
        message.append(" Final Clipping " + l.lineStatus());
        showClipping_Line(l, b, message);
    }
}

int main(){
    Boundary b = {100, 300, 100, 400};
    Line l = {Point(50, 200), Point(450, 100)};
```

```
    /*Graph initialisations*/
    int graphMode, graphDriver = DETECT;
    initgraph(&graphDriver, &graphMode, NULL);
    setbkcolor(WHITE);
    setcolor(BLACK);
    setfontcolor(BLACK);
    /**/

    cohenSuther_clipping(b, l);

    delay(100000);
    closegraph();

    return 0;
}
```

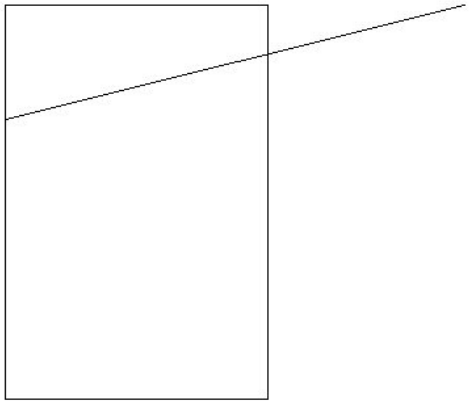
OUTPUT:

1)



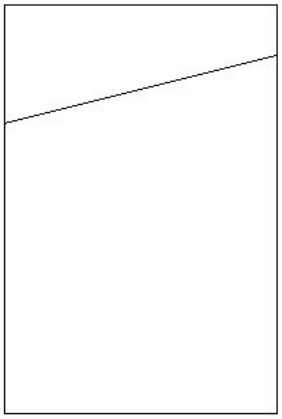
2)

Some part of line is inside (100,187) to (450,100)



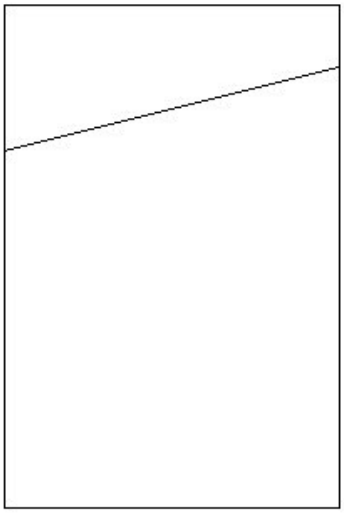
3)

Some part of line is inside (100,187) to (300,137)



4)

Line is completely inside Final Clipping (100,187) to (300,137)



11. Write a program to clip a line using Liang Barsky Algorithm

Description:

In computer graphics, the Liang–Barsky algorithm (named after You-Dong Liang and Brian A. Barsky) is a line clipping algorithm. The Liang–Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clipping window. With these intersections it knows which portion of the line should be drawn. This algorithm is significantly more efficient than Cohen–Sutherland.The idea of the Liang-Barsky clipping algorithm is to do as much testing as possible before computing line intersections.

Program:

```
#include <bits/stdc++.h>
#include <graphics.h>
using namespace std;

/* Algorithm for line clipping using Liang-Barsky algorithm.  Clip a line l given a boundary b*/

typedef struct Boundary{
   int x_min, x_max, y_min, y_max;
}Boundary;

typedef struct Point{
   int x, y;
   Point (){}
   Point(int a, int b) : x(a), y(b){}
}Point;

typedef struct Line{
   Point A, B;
   string lineStatus(){
         return "(" + to_string(A.x)  + "," + to_string(A.y) + ") to (" + to_string(B.x) + "," +
to_string(B.y) + ")\n";
   }
}Line;

void liangBarsky_clipping(Boundary &b, Line &l){
   //Perform liang Barsky clipping for the given line and boundary
```

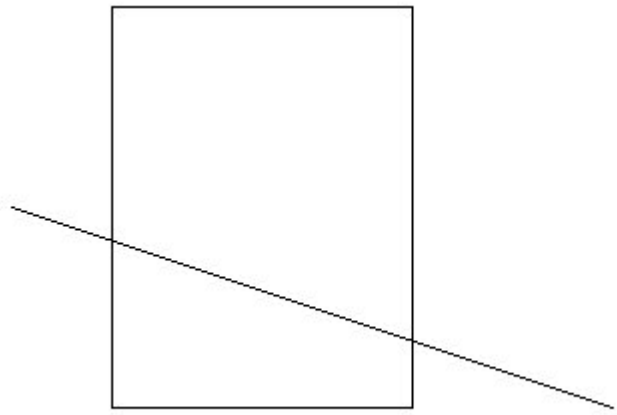
```
    rectangle(b.x_min, b.y_max, b.x_max, b.x_min);
    line(l.A.x, l.A.y, l.B.x, l.B.y);
    sleep(2);
    /*
            A) Parametric line equation for (x1, x2) and (y1, y2) as f(t)
               x(t) = x1 + t(x2 - x1) and y(t) = y1 + t(y2 - y1)
               b.xmin <= x <= b.xmax  and  b.ymin <= y <= b.ymax
               b.xmin <= x1 + t(dx) <= b.xmax
               b.ymin <= y1 + t(dy) <= b.ymax
                  OR
               -t(dx) <= x1 - b.xmin
               t(dx)  <= b.xmax - x1
               -t(dy) <= y1 - b.ymin
               t(dy)  <= b.ymax - y1

               t(p(k)) <= q(k) where k = 1, 2, 3, 4.. Rearranging,
               p1 = -dx, q1 = x1 - b.xmin
               p2 = dx , q2 = b.xmax - x1
               p3 = -dy, q3 = y1 - b.ymin
               p4 = dy,  q4 = b.ymax - y1

            B) Assign t1 = 0, t2 = 1
               if p(k) (i.e dx or dy) == 0: then line is || to boundary
               if q(k) < 0, discard line as it is outside
               if p(k) != 0:
                  if p(k) < 0: partially in. Find st1rting point t1 = max(0, q(k)/p(k))
                  else: Find t2 (ending point) = min(1, q(k)/p(k))
               if t1 > t2: reject
               else: find new set of (x, y)
                  x = x1 + t(dx) and y = y1 + t(dy)
    */
    int dx = l.B.x - l.A.x, dy = l.B.y - l.A.y;
    vector<int> P {-dx, dx, -dy, dy};
    vector<int> Q {l.A.x - b.x_min, b.x_max - l.A.x, l.A.y - b.y_min, b.y_max - l.A.y};

    for (int i = 0; i < P.size(); i++)
        cout << "\n P["<<i<<"] : " << P[i] << " and Q["<<i<<"] : " << Q[i] << " Q/P: " << (Q[i] *
1.0)/P[i];
    cout << endl;

    float t1 = 0.0, t2 = 1.0;
    for (int k = 0; k < P.size(); k++){
       if (P[k] == 0 && Q[k] < 0){
```

```
            cout << "\n Line is outside of clipping window. Rejecting Line\n";
            return;
        }
        if (P[k] < 0)
            t1 = max(t1, (Q[k] * 1.0)/P[k]);
        else
            t2 = min(t2, (Q[k] * 1.0)/P[k]);
    }
    cout << "\n t1: " << t1 << endl;
    cout << "\n t2: " << t2 << endl;

    if (t1 > t2){ //reject line
        cout << "\n Line is completely outside the window. Rejecting line";
        return;
    }

    Point newA(l.A.x + t1 * dx, l.A.y + t1 * dy);
    Point newB(l.A.x + t2 * dx, l.A.y + t2 * dy);
    Line clippedLine = {newA, newB};

    fprintf(stdout, "\n New x1: (%d, %d)", newA.x, newA.y);
    fprintf(stdout, "\n New x2: (%d, %d)", newB.x, newB.y);
    fprintf(stdout, "\n Clipped line: %s" ,clippedLine.lineStatus().c_str());

    cleardevice();
    rectangle(b.x_min, b.y_max, b.x_max, b.x_min);
    line(clippedLine.A.x, clippedLine.A.y, clippedLine.B.x, clippedLine.B.y);
    cout << "\n CLIPPING FINISHED";
}

int main(){
    Boundary b = {100, 250, 100, 300};
    Line l = {Point(50, 200), Point(350, 300)};

    /*Graph initialisations*/
    int graphMode, graphDriver = DETECT;
    initgraph(&graphDriver, &graphMode, NULL);
    setbkcolor(WHITE);
    setcolor(BLACK);
    /**/
```

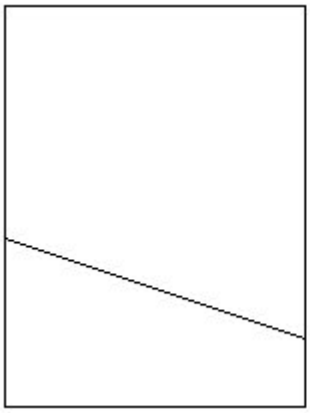
```
    liangBarsky_clipping(b, l);
    delay(100000);
    closegraph();
    return 0;
}
```

OUTPUT:

A: BEFORE CLIPPING



B: AFTER CLIPPING



```
P[0] : -300 and Q[0] : -50 Q/P: 0.166667
P[1] : 300 and Q[1] : 200 Q/P: 0.666667
P[2] : -100 and Q[2] : 100 Q/P: -1
P[3] : 100 and Q[3] : 100 Q/P: 1

t1: 0.166667

t2: 0.666667

New x1: (100, 216)
New x2: (250, 266)
Clipped line: (100,216) to (250,266)
```

12. Write a program to draw cubic Bezier curves

Description:

A Bézier curve is a parametric curve frequently used in computer graphics and related fields. Generalizations of Bézier curves to higher dimensions are called Bézier surfaces, of which the Bézier triangle is a special case. In vector graphics, Bézier curves are used to model smooth curves that can be scaled indefinitely. "Paths", as they are commonly referred to in image manipulation programs, are combinations of linked Bézier curves. Paths are not bound by the limits of rasterized images and are intuitive to modify.

Some terminology is associated with these parametric curves. We have

$$\mathbf{B}(t) = \sum_{i=0}^{n} b_{i,n}(t)\mathbf{P}_i, \quad 0 \le t \le 1$$

where the polynomials

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \ldots, n$$

In this program, four control points are inputted from the user. The cubicBezier class instantiates an object with these four control points. Finally, the curve is depicted.

Program:

```
#include <bits/stdc++.h>
#include <graphics.h>
using namespace std;

/*A program to implement cubic Bezier curves. (deg =3) So number of control points : 4
  Input : Four Control Points . Output: Bezier curves
*/
typedef struct Point{
   int x, y;
   Point(){}
   Point(int a, int b) : x(a), y(b){}
   friend ostream & operator << (ostream &c, Point &a){
      c << " Point : (" << a.x << ", " << a.y << ")";
      return c;
   }
```

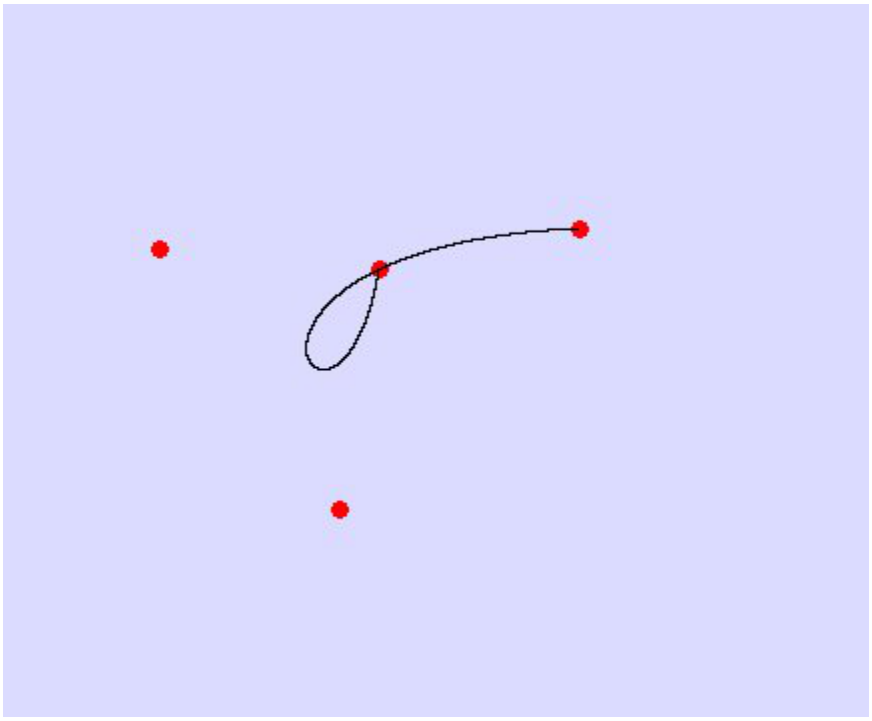
```
    void operator =(Point c){
        x = c.x, y = c.y;
    }
}Point;

class Cubic_Bezier{
    //A class that represents a cubic bezier curve
    // f(t) = (1 - t)^3 * a + 3(1 - t)^2 * t * b + 3*(1 - t)*t*t*c + t^3 * d
    Point a, b, c, d;
    int graphMode, graphDriver = DETECT;

    Point func(double t){
        //Given a parametric t, find the new coordinates from the equation
        double x = pow(1 - t, 3) * a.x + 3 * pow(1 - t, 2) * t * b.x +
                3 * (1 - t) * t * t * c.x + pow(t, 3) * d.x;
        double y = pow(1 - t, 3) * a.y + 3 * pow(1 - t, 2) * t * b.y +
                3 * (1 - t) * t * t * c.y + pow(t, 3) * d.y;
        return Point(x, y);
    }

    public:
        Cubic_Bezier(Point &pA, Point &pB, Point &pC, Point &pD){
            //Initialize the graphic utilities annd control points
            a = pA, b = pB, c = pC, d = pD;
            initgraph(&graphDriver, &graphMode, NULL);
            setcolor(RED);
            setbkcolor(LIGHTGRAY);
        }

        ~Cubic_Bezier(){
            delay(10000);
            closegraph();
        }

        void draw_Curve(){
            //Draw the Bezier curve and Plot Control points in red (filled preferably):
            circle(a.x, a.y, 4); floodfill(a.x, a.y, RED);
            circle(b.x, b.y, 4); floodfill(b.x, b.y, RED);
            circle(c.x, c.y, 4); floodfill(c.x, c.y, RED);
            circle(d.x, d.y, 4); floodfill(d.x, d.y, RED);
```

```
        int iter = 0;
        for (double t = 0.0; t <= 1; iter++, t += 0.00005){
            Point newP = func(t);
            if (iter % 2000 == 0) //view any 10 iterations
                cout << "\n f(" << t << "): " << newP;
            putpixel(newP.x, newP.y, BLACK);
        }
    }
};

int main(){
    Point A, B, C, D;
    cout << "\n CUBIC BEZIER CURVES (4 CONTROL POINTS)\n";
    cout << "\n ENTER FOUR X VALUES: \t";
    cin >> A.x >> B.x >> C.x >> D.x;
    cout << "\n ENTER FOUR Y VALUES: \t";
    cin >> A.y >> B.y >> C.y >> D.y;

    Cubic_Bezier cb(A, B, C, D);
    cb.draw_Curve();
    return 0;
}
```

OUTPUT:

```
aman@aman ~/Desktop/prog/Graphics/3-31 BEZIER $ ./bezier

 CUBIC BEZIER CURVES (4 CONTROL POINTS)

 ENTER FOUR X VALUES:   220 200 110 320

 ENTER FOUR Y VALUES:   230 350 220 210

 f(0):  Point : (220, 230)
 f(0.1):  Point : (212, 258)
 f(0.2):  Point : (202, 274)
 f(0.3):  Point : (193, 280)
 f(0.4):  Point : (186, 277)
 f(0.5):  Point : (183, 268)
 f(0.6):  Point : (188, 255)
 f(0.7):  Point : (202, 241)
 f(0.8):  Point : (227, 227)
 f(0.9):  Point : (265, 216)
```

BEZIER CURVE:



13. Write a program to demonstrate simple animation.

Description:

Animation is the process of making the illusion of motion and change by means of the rapid display of a sequence of static images that minimally differ from each other. The illusion—as in motion pictures in general—is thought to rely on the phi phenomenon. Animators are artists who specialize in the creation of animation.

Animation can be recorded with either analogue media, a flip book, motion picture film, video tape, digital media, including formats with animated GIF, Flash animation and digital video. To display animation, a digital camera, computer, or projector are used along with new technologies that are produced.

Animation creation methods include the traditional animation creation method and those involving stop motion animation of two and three-dimensional objects, paper cutouts, puppets and clay figures. Images are displayed in a rapid succession, usually 24, 25, 30, or 60 frames per second.

Program:

```
#include <bits/stdc++.h>
#include <graphics.h>
using namespace std;

int graphMode, graphDriver = DETECT;
double coeffRestitution = 0.8;

// A Program to demonstrate simple animation
// Program demonstrates a ball falling from top of the screen to the base
// The bouncing ball falls in accordance with the gravity

void initialize(){
   initgraph(&graphDriver, &graphMode, NULL);
   setbkcolor(LIGHTGRAY);
   setcolor(RED);
}
```

```
void plotBall(int h, int k, int radius){
   circle(h, k, radius);
   floodfill(h, k, RED);
   delay(2);
   cleardevice();
}

int getNextHeight(int height){
   //find next height upto which the ball would bounce after falling from h
   return int(pow(coeffRestitution, 2) * height);
}

int main(){
   initialize();
   int x_c = getmaxx()/2 - 10;
   int radius = 15, maxy = getmaxy();
   int base = maxy - radius;

   int height = maxy - radius;
   while (maxy - height <= base - radius){
      fprintf(stdout, "\n Falling from height : %d to %d", height, getNextHeight(height));
      for (int y_c = getmaxy() - height; y_c != base; y_c += 1)
         plotBall(x_c, y_c, radius);

      height = getNextHeight(height);
      for (int y_c = base; y_c != getmaxy() - height; y_c --)
         plotBall(x_c, y_c, radius);
   }

   delay(10000);
   closegraph();

   return 0;
}
```

OUTPUT: