COMPUTER GRAPHICS

LAB PRACTICALS RECORD

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR – 144011, PUNJAB (INDIA)

Submitted To:

Ms. Jagriti Kaur
Asst. Professor
Department of CSE

Submitted By:

Nikhil Bansal
13103011
6th Semester

PROGRAM 1

DDA LINE ALGORITHM

Description:

In computer graphics, a digital differential analyzer (DDA) is hardware or software used for linear interpolation of variables over an interval between start and end point. DDAs are used for rasterization of lines, triangles and polygons. In its simplest implementation, the DDA algorithm interpolates values in interval by computing for each x_i the equations

$$x_i = x_{i-1} + 1/m$$
$$y_i = y_{i-1} + m$$
$$\Delta x = x_{end} - x_{start} \text{ and } \Delta y = y_{end} - y_{start} \text{ and } m = \Delta y/\Delta x.$$

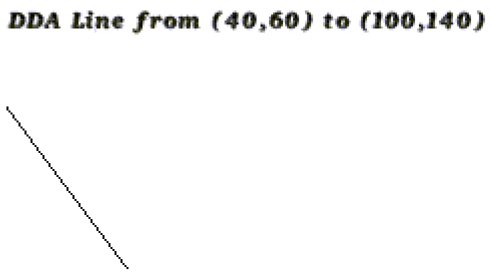
Program:

```
#include<graphics.h>
#include<stdio.h>

int main(int argc,char *argv[])
{
  if(argc<5){
    printf("Enter coordinates of end points of line on commandine\n");
    return -1;
  }

  //coordinates output file
  FILE *coordinates=fopen("coordinates", "w");

  //commandline input
  int i,j,x1,x2,y1,y2;
  float currx,curry;
  x1=atoi(argv[1]);
  y1=atoi(argv[2]);
  x2=atoi(argv[3]);
  y2=atoi(argv[4]);

  //if coordinates are not in increasing order of x then make them
  if(x1>x2){
    int temp=x1;
    x1=x2;
    x2=temp;
    temp=y1;
    y1=y2;
    y2=temp;
```

```
}

//graphics initialise
int gd = DETECT,gm;
initgraph(&gd,&gm,NULL);
setbkcolor(WHITE);
setcolor(BLACK);

//find the slope
float m=((float)y2-y1)/(x2-x1);

if(m<=1&&m>=-1){
  putpixel(x1,y1,getcolor());
  currx=x1;
  curry=y1;
  while(currx!=x2){
    currx+=1;
    curry=curry+m;
    fprintf(coordinates, "%d %d\n",(int)currx,(int)curry);
    putpixel((int)currx,(int)curry,getcolor());
  }
}

if(m>1||m<-1){
  if(m>1){
    putpixel(x1,y1,getcolor());
    currx=x1;
    curry=y1;
    while(curry!=y2){
      curry+=1;
      currx=currx+1/m;
      fprintf(coordinates,"%d %d\n",(int)currx,(int)curry);
      putpixel((int)currx,(int)curry,getcolor());
    }
  }
  else{
    putpixel(x2,y2,getcolor());
    currx=x2;
    curry=y2;
    while(curry!=y1){
      curry+=1;
      currx=currx+1/m;
      fprintf(coordinates, "%d %d\n",(int)currx,(int)curry);
      putpixel((int)currx,(int)curry,getcolor());
    }
  }
```

```
   }

   delay(5000);
   closegraph();
   return 0;
}
```

Output:



DDA Line from (40,60) to (100,140)

PROGRAM 2

BRESNHAM'S LINE ALGORITHM

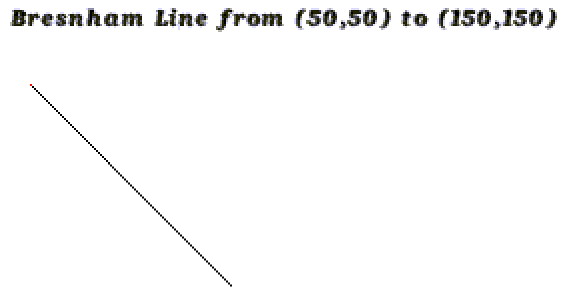
Description:

The Bresenham's line algorithm is an algorithm that determines the points of an *n*-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. It is commonly used to draw lines on a computer screen, as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is one of the earliest algorithms developed in the field of computer graphics. An extension to the original algorithm may be used for drawing circles.

Program:

```
#include<graphics.h>
#include<stdio.h>

// absolute i.e mod of x
int abs(int x){
  if(x<0)
    return -x;
  else
    return x;
}

// bresnham line algo used to draw line
// works only for |m|<1
int bresnhamLine(int x1, int y1, int x2, int y2){
  int p_curr,currx,curry;

  // coordinates output file
  FILE *coordinates=fopen("coordinates", "w");

  //slope
  float m=((float)y2-y1)/(x2-x1);
  int dx=abs(x2-x1);
  int dy=abs(y2-y1);
  putpixel(x1, y1, getcolor());

  // algorithm
  if(m<=1 && m>=-1){
    currx=x1;
    curry=y1;
    p_curr=2*dy-dx;
    putpixel(x1,y1,RED);
    if(m>=0){
      for(currx=x1+1;currx<=x2;currx++){
        if(p_curr>=0){
          curry++;
          p_curr=p_curr+2*dy-2*dx;
        }
        else{
          p_curr=p_curr+2*dy;
        }
        fprintf(coordinates,"%d %d\n",currx,curry);
        putpixel(currx, curry, getcolor());
      }
    }
    else{
      for(currx=x1+1;currx<=x2;currx++){
        if(p_curr>=0){
          curry--;
          p_curr=p_curr+2*dy-2*dx;
        }
        else{
          p_curr=p_curr+2*dy;
        }
        fprintf(coordinates,"%d %d\n",currx,curry);
        putpixel(currx,curry,RED);
        putpixel(currx+5, curry+5, getcolor());
      }
    }
  }

  fclose(coordinates);
  return 0;
}

int main(int argc,char *argv[]){
  // command line arguments check
  if(argc<5){
    printf("Enter coordinates of end points of line on commandine\n");
    return -1;
  }
  // commandline input
  int x1,x2,y1,y2;
  x1=atoi(argv[1]);
  y1=atoi(argv[2]);
  x2=atoi(argv[3]);
  y2=atoi(argv[4]);

  //if coordinates are not in increasing order of x then make them
  if(x1>x2){
    int temp=x1;
    x1=x2;
    x2=temp;
    temp=y1;
    y1=y2;
    y2=temp;
  }

  // Initialise graphics
  int gd = DETECT,gm;
  initgraph(&gd,&gm,NULL);
  setbkcolor(WHITE);
  setcolor(BLACK);

  // draw line
  bresnhamLine(x1,y1,x2,y2);

  // delay to able to view graphics
        delay(5000);
  closegraph();
        return 0;
}
```

Output:



PROGRAM 3

TRIGNOMETRIC CIRCLE

Description:

It is the basic algorithm used to draw circle. In this algo, we basically find the coordinates by using the trigonometry formulas. We find x and y coordinates by :

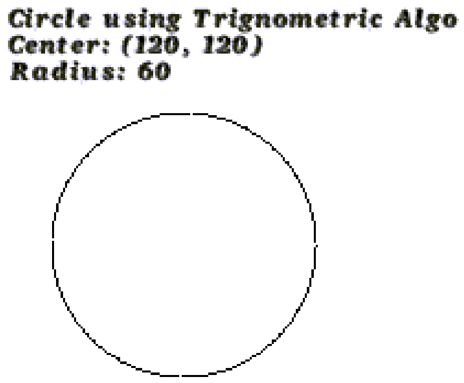
$$x = r * \cos(angle)$$
$$y = r * \sin(angle)$$

Program:

```
#include<graphics.h>
#include<math.h>
#include<stdio.h>

// draw the circle with given integer center and radius
int trignometricCircle(int x,int y,int radius){
   float curr_x,curr_y;
   int angle;
   FILE *coordinates=fopen("coordinates", "w");

   // algo
   for(angle=0;angle<360;angle++){
      curr_x=x+cos((float)angle/180*3.14)*radius;
      curr_y=y+sin((float)angle/180*3.14)*radius;
      putpixel((int)curr_x,(int)curr_y,getcolor());
      fprintf(coordinates, "%d %d\n", (int)curr_x, (int)curr_y);
   }
   return 0;
}

int main(int argc,char *argv[]){

   //command-line parameters check
   if(argc<3){
      printf("Enter 3 arguments on commandine\n");
      return 0;
   }

   //graphics initialisation
   int gd = DETECT,gm;
   initgraph(&gd,&gm,NULL);
   setbkcolor(WHITE);
   setcolor(BLACK);

   //get the center and radius
   int x,y,radius;
   x=atoi(argv[1]);
   y=atoi(argv[2]);
```

```
  radius=atoi(argv[3]);

  //Draw the circle using Trignometric algo
  trignometricCircle(x,y,radius);

  //delay so as to view the screen
  delay(5000);
         return 0;
}
```

Output:



```
Circle using Trignometric Algo
Center: (120, 120)
Radius: 60
```

PROGRAM 4

MID POINT CIRCLE

Description:

In computer graphics, the midpoint circle algorithm is an algorithm used to determine the points needed for drawing a circle. Bresenham's circle algorithm is derived from the midpoint circle algorithm. The algorithm can be generalized to conic sections. It is more efficient than trigonometric circle algorithm as it doesn’t use any trigonometric functions.
The algorithm is related to work by Pitteway and Van Aken.

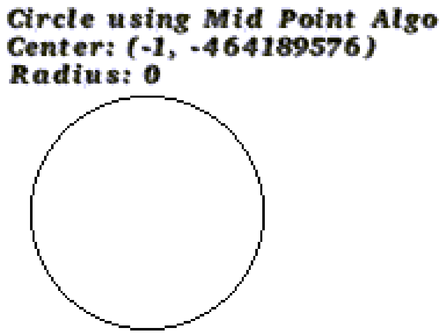
Program:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>

int midPointCircle(float x,float y,float radius){
        //coordinates output file
        FILE *coordinates=fopen("coordinates", "w");

        float pinit,pcurr;
        int curr_x,curr_y;

        // calculate the initial decision parameter
        if(floor(radius)-radius==0)
                pinit=1-radius;
        else
                pinit=5.00/4-radius;

        // initialisations
        curr_x=0;
        curr_y=floor(radius);
        pcurr=pinit;

        // operate while loop until x<y
        while(curr_x<=curr_y){
                // output points
                putpixel((int)(curr_x+x),(int)(curr_y+y),getcolor());
                putpixel((int)(-curr_x+x),(int)(curr_y+y),getcolor());
                putpixel((int)(curr_x+x),(int)(-curr_y+y),getcolor());
                putpixel((int)(-curr_x+x),(int)(-curr_y+y),getcolor());
                putpixel((int)(curr_y+y),(int)(curr_x+x),getcolor());
                putpixel((int)(-curr_y+y),(int)(curr_x+x),getcolor());
                putpixel((int)(curr_y+y),(int)(-curr_x+x),getcolor());
```

```
			putpixel((int)(-curr_y+y),(int)(-curr_x+x),getcolor());
			fprintf(coordinates,"%d %d\n",(int)(curr_x+x),(int)(curr_y+y));
			fprintf(coordinates,"%d %d\n",(int)(-curr_x+x),(int)(curr_y+y));
			fprintf(coordinates,"%d %d\n",(int)(curr_x+x),(int)(-curr_y+y));
			fprintf(coordinates,"%d %d\n",(int)(-curr_x+x),(int)(-curr_y+y));
			fprintf(coordinates,"%d %d\n",(int)(curr_y+y),(int)(curr_x+x));
			fprintf(coordinates,"%d %d\n",(int)(-curr_y+y),(int)(curr_x+x));
			fprintf(coordinates,"%d %d\n",(int)(curr_y+y),(int)(-curr_x+x));
			fprintf(coordinates,"%d %d\n",(int)(-curr_y+y),(int)(-curr_x+x));

			// algo
			if(pcurr<0){
				curr_x+=1;
				pcurr=pcurr+2*curr_x+1;
			}
			else{
				curr_x+=1;
				curr_y-=1;
				pcurr=pcurr+2*curr_x+1-2*curr_y;
			}
		}

		// close the output file
		fclose(coordinates);
		return 0;
}

int main(int argc,char *argv[]){

   //command-line parameters check
   if(argc<3){
      printf("Enter 3 arguments on commandine\n");
      return 0;
   }

		//get the center and radius
   float x,y,radius;
   x=atoi(argv[1]);
   y=atoi(argv[2]);
   radius=atof(argv[3]);

   // check if x and y are greater than radius else pixel out of range will be there
   if(x<radius||y<radius){
			printf("Circle cannot be displayed\nAs x and y are less than radius so there will be pixel out of
range.\n");
			return 0;
```

```
    }

    //graphics initialisation
    int gd = DETECT,gm;
    initgraph(&gd,&gm,NULL);
    setbkcolor(WHITE);
            setcolor(BLACK);

    //Draw the circle using Trignometric algo
    midPointCircle(x,y,radius);

    //delay so as to view the screen
    delay(5000);
            return 0;
}
```

Output:



```
Circle using Mid Point Algo
Center: (-1, -464189576)
Radius: 0
```

PROGRAM 5

TRIGNOMETRIC ELLIPSE

Description:

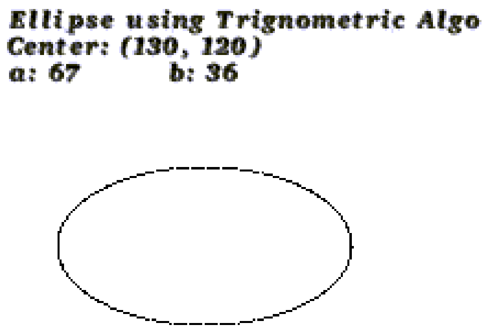
It is the basic algorithm used to draw circle. In this algo, we basically find the coordinates by using the trigonometry formulas. We find x and y coordinates by :

x = a * cos(angle)
y = b * sin(angle)

Program:

```
#include<graphics.h>
#include<math.h>
#include<stdio.h>

// draw the circle with given integer center and axes
int trignometricEllipse(int x,int y,int a,int b){
  float curr_x,curr_y;
  int angle;
  FILE *coordinates=fopen("coordinates", "w");

  // algo
  for(angle=0;angle<360;angle++){
    curr_x=x+a*cos((float)angle/180*3.14);
    curr_y=y+b*sin((float)angle/180*3.14);
    putpixel((int)curr_x,(int)curr_y,getcolor());
    fprintf(coordinates, "%d %d\n", (int)curr_x, (int)curr_y);
  }

  fclose(coordinates);
  return 0;
}

int main(int argc,char *argv[]){

  //command-line parameters check
  if(argc<4){
    printf("Enter 4 arguments on commandine\n");
    return 0;
  }

  //get the center and radius
  int x,y,a,b;
  x=atoi(argv[1]);
  y=atoi(argv[2]);
  a=atoi(argv[3]);
  b=atoi(argv[4]);

  // check for pixel out of range
  if(x<a||y<b){
    printf("Enter center of ellipse such that center points are less than a and b.\nElse therer will be pixel out of
range.\n");
    return 0;
  }

  //graphics initialisation
  int gd = DETECT,gm;
  initgraph(&gd,&gm,NULL);
  setbkcolor(WHITE);
  setcolor(BLACK);

  //Draw the ellipse using Trignometric algo
  trignometricEllipse(x,y,a,b);

  //delay so as to view the screen
  delay(5000);
  closegraph();
        return 0;
}
```

Output:



```
Ellipse using Trignometric Algo
Center: (130, 120)
a: 67        b: 36
```

PROGRAM 6

MID POINT ELLIPSE

Description:

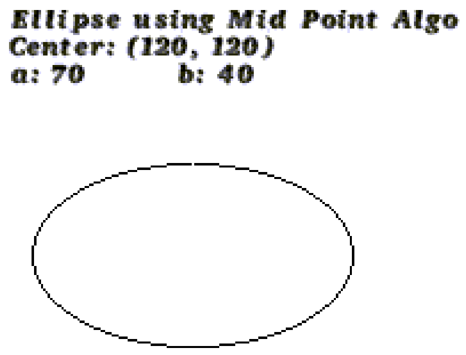
RSA In computer graphics, the midpoint ellipse algorithm is an algorithm used to determine the points needed for drawing a ellipse. The algorithm can be generalized to conic sections. It is more efficient than trigonometric ellipse algorithm as it doesn't use any trigonometric functions. It considers the distance of the midpoint to the point on the ellipse and takes the nearest point.
The algorithm is related to work by Pitteway and Van Aken.

Program:

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

// draw the circle with given integer center and axes
int midPointEllipse(int x,int y,int a,int b){
        int curr_x,curr_y,pcurr;
  FILE *coordinates=fopen("coordinates", "w");

  // initialisations
  curr_x=0;
  curr_y=b;
  pcurr=b*b-a*a*b+a*a/4;
  putpixel(x+curr_x, y+curr_y, getcolor());

  // region 1
  while(2*b*b*curr_x<2*a*a*curr_y){
    if(pcurr<0){
      curr_x++;
      pcurr=pcurr+2*b*b*curr_x+b*b;
    }
    else{
      curr_x++;
      curr_y--;
      pcurr=pcurr+2*b*b*curr_x-2*a*a*curr_y+b*b;
    }
    putpixel(x+curr_x, y+curr_y, getcolor());
    putpixel(x-curr_x, y-curr_y, getcolor());
    putpixel(x+curr_x, y-curr_y, getcolor());
    putpixel(x-curr_x, y+curr_y, getcolor());
    fprintf(coordinates, "%d %d\n", curr_x, curr_y);
  }

  // region 2
  pcurr=b*b*(curr_x+0.5)*(curr_x+0.5)+a*a*(curr_y-1)*(curr_y-1)-a*a*b*b;
  while(curr_y!=0){
    if(pcurr>0){
      curr_y--;
      pcurr=pcurr-2*a*a*curr_y+a*a;
    }
    else{
      curr_x++;
      curr_y--;
      pcurr=pcurr+2*b*b*curr_x-2*a*a*curr_y+a*a;
    }
    putpixel(x+curr_x, y+curr_y, getcolor());
    putpixel(x-curr_x, y-curr_y, getcolor());
    putpixel(x-curr_x, y+curr_y, getcolor());
    putpixel(x+curr_x, y-curr_y, getcolor());
    fprintf(coordinates, "%d %d\n", curr_x, curr_y);
  }

  fclose(coordinates);
        return 0;
}

int main(int argc,char *argv[]){

  //command-line parameters check
  if(argc<4){
    printf("Enter 4 arguments on commandine\n");
    return 0;
  }

  //get the center and radius
  int x,y,a,b;
  x=atoi(argv[1]);
  y=atoi(argv[2]);
  a=atoi(argv[3]);
  b=atoi(argv[4]);

  // check for pixel out of range
  if(x<a||y<b){
    printf("Enter center of ellipse such that center points are less than a and b.\nElse therer will be pixel out of
range.\n");
    return 0;
  }

  //graphics initialisation
  int gd = DETECT,gm;
  initgraph(&gd,&gm,NULL);
  setbkcolor(WHITE);
  setcolor(BLACK);

  //Draw the ellipse using Trignometric algo
  midPointEllipse(x,y,a,b);

  //delay so as to view the screen
  delay(5000);
```

```
        return 0;
}
```

Output:



```
Ellipse using Mid Point Algo
Center: (120, 120)
a: 70        b: 40
```

PROGRAM 7

2D TRANSFORMATIONS

Description:

The Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.
Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Program:

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

// x and y coordinates of a point
class point{
public:
	int x,y;
	point(){
		x=0;
		y=0;
	}
	point(int x,int y){
		this->x=x;
		this->y=y;
	}
};

// x and y are translations in x and y directions
class translation{
public:
	int x,y;
	translation(){
		x=0;
		y=0;
	}
	translation(int x,int y){
		this->x=x;
		this->y=y;
	}
};
```

```
// angle is the angle of rotation
class rotation{
public:
        float angle;
        rotation(){
                angle=0;
        }
        rotation(int angle){
                this->angle=angle;
        }
};

// x and y are scaling in x and y directions
class scale{
public:
        int x,y;
        scale(){
                x=0;
                y=0;
        }
        scale(int x, int y){
                this->x=x;
                this->y=y;
        }
};

// xabout signifies if shear is in x direction
// x is the amount of shear about given yAxis
class shear{
public:
        int x,y,xAxis,yAxis;
        bool xAbout,yAbout;
        shear(){
                x=0;
                y=0;
        }
        shear(int x, int y, int xAxis, int yAxis, bool xAbout, bool yAbout){
                this->x=x;
                this->y=y;
                this->xAxis=xAxis;
                this->yAxis=yAxis;
                this->xAbout=xAbout;
                this->yAbout=yAbout;
        }
};
```

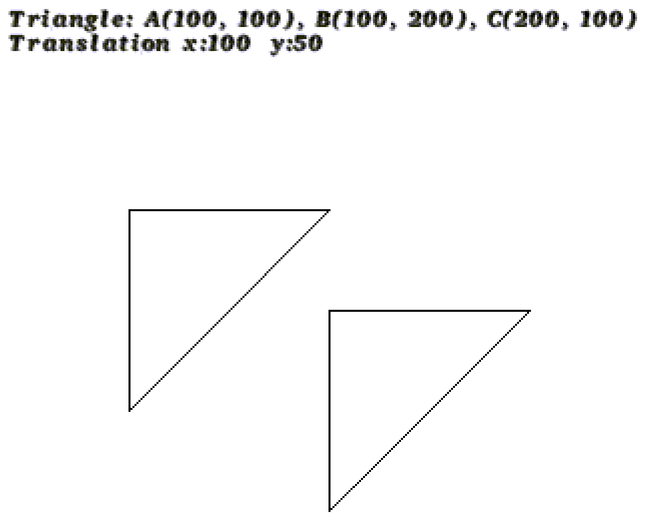
```
// xAbout and yAbout tells to reflect about x or y axis
class reflection{
public:
	bool xAbout, yAbout;
	reflection(){
		xAbout=0;
		yAbout=0;
	}
	reflection(bool xAbout, bool yAbout){
		this->xAbout=xAbout;
		this->yAbout=yAbout;
	}
};

point translatePoint(point p, translation t){
	p.x+=t.x;
	p.y+=t.y;
	return p;
}

point rotatePoint(point p, rotation r){
	point result;
	result.x=abs((p.x)*cos(r.angle)-(p.y)*sin(r.angle));
	result.y=abs((p.x)*sin(r.angle)+(p.y)*cos(r.angle));
	return result;
}

point scalePoint(point p, scale s){
	p.x=(s.x)*(p.x);
	p.y=(s.y)*(p.y);
	return p;
}

point shearPoint(point p, shear sh){
	if(sh.xAbout){
		p.x = (p.x) + (sh.x) * ((p.y)-(sh.yAxis));
	}
	else if(sh.yAbout){
		p.y = (p.y) + (sh.y) * ((p.x)-(sh.xAxis));
	}
	return p;
}

point reflectPoint(point p, reflection rf){
	if(rf.xAbout){
		p.y = -1*p.y;
	}
	if(rf.yAbout){
		p.x = -1*p.x;
	}
	return p;
}

int plotPoint(point p){
	putpixel(p.x, p.y, getcolor());
	return 0;
}

int plotTriangle(point a, point b, point c){
	line(a.x, a.y, b.x, b.y);
	line(b.x, b.y, c.x, c.y);
	line(c.x, c.y, a.x, a.y);
	return 0;
}

int printPoint(point p){
	printf("(%d, %d)\n",p.x,p.y);
	return 0;
}

int printTriangle(point a, point b, point c){
	printPoint(a);
	printPoint(b);
	printPoint(c);
	return 0;
}

int main(int argc,char *argv[]){
	FILE *input=fopen("input", "r");

	// initialisations
	int temp,temp1;
	point a,b,c,aFinal,bFinal,cFinal;
	translation t;
	rotation r;
	scale s;
	shear sh;
	reflection rf;

	// input coordinates of triangle
	fscanf(input,"%d%d",&a.x,&a.y);
	fscanf(input,"%d%d",&b.x,&b.y);
	fscanf(input,"%d%d",&c.x,&c.y);

	// translation, rotation and scaling
	fscanf(input,"%d%d",&t.x,&t.y);
	fscanf(input,"%f",&r.angle);
	fscanf(input,"%d%d",&s.x,&s.y);

	// shear , temp=0 implies shear in x else shear in y
	fscanf(input,"%d",&temp);
	if(temp==0){
		sh.xAbout=1;
		fscanf(input,"%d%d", &sh.x, &sh.yAxis);
	}
	else{
		sh.yAbout=1;
		fscanf(input,"%d%d", &sh.y, &sh.xAxis);
	}

	// reflection temp and temp1 signify if reflection about x or y
	fscanf(input, "%d%d", &temp,&temp1);
	if(temp)
		rf.xAbout=1;
	if(temp1)
		rf.yAbout=1;

	// graphics initialisation
	int gd = DETECT,gm;
	initgraph(&gd,&gm,NULL);
	setbkcolor(WHITE);
	setcolor(BLACK);
	setfontcolor(BLACK);

	// translation
	printf("\tTriangle: A(%d, %d), B(%d, %d), C(%d, %d)\n\tTranslation x:%d  y:%d", a.x, a.y, b.x, b.y,
c.x, c.y, t.x, t.y);
	plotTriangle(a,b,c);
	aFinal=translatePoint(a,t);
	bFinal=translatePoint(b,t);
	cFinal=translatePoint(c,t);
	sleep(3);
	plotTriangle(aFinal,bFinal,cFinal);

	// rotation
	sleep(3);
	cleardevice();
	printf("\n\tTriangle: A(%d, %d), B(%d, %d), C(%d, %d)\n\tRotation Angle: %f in radians", a.x, a.y, 
b.x, b.y, c.x, c.y, r.angle);
	plotTriangle(a,b,c);
	aFinal=rotatePoint(a,r);
	bFinal=rotatePoint(b,r);
	cFinal=rotatePoint(c,r);
	sleep(3);
	plotTriangle(aFinal,bFinal,cFinal);

	// scaling
	sleep(3);
	cleardevice();
	printf("\n\tTriangle: A(%d, %d), B(%d, %d), C(%d, %d)\n\tScale x:%d  y:%d", a.x, a.y, b.x, b.y, c.x, 
c.y, s.x, s.y);
	plotTriangle(a,b,c);
	aFinal=scalePoint(a,s);
	bFinal=scalePoint(b,s);
	cFinal=scalePoint(c,s);
	sleep(3);
	plotTriangle(aFinal,bFinal,cFinal);

	// shear
	sleep(3);
	cleardevice();
	if(sh.xAbout)
		printf("\n\tTriangle: A(%d, %d), B(%d, %d), C(%d, %d)\n\tShear about y: %d by amount: 
%d", a.x, a.y, b.x, b.y, c.x, c.y, sh.yAxis, sh.x);
	else
		printf("\n\tTriangle: A(%d, %d), B(%d, %d), C(%d, %d)\n\tShear about x: %d by amount: 
%d", a.x, a.y, b.x, b.y, c.x, c.y, sh.xAxis, sh.y);
	plotTriangle(a,b,c);
	aFinal=shearPoint(a,sh);
	bFinal=shearPoint(b,sh);
	cFinal=shearPoint(c,sh);
	sleep(3);
	plotTriangle(aFinal, bFinal, cFinal);

	// reflection
	// sleep(3);
	// cleardevice();
	aFinal=reflectPoint(a,rf);
	bFinal=reflectPoint(b,rf);
	cFinal=reflectPoint(c,rf);
	// sleep(3);
	// plotTriangle(aFinal, bFinal, cFinal);
```

```
        //delay so as to view the screen and close the graph
        delay(5000);
        closegraph();
        fclose(input);

        printTriangle(aFinal, bFinal, cFinal);
        return 0;
}
```

Output:

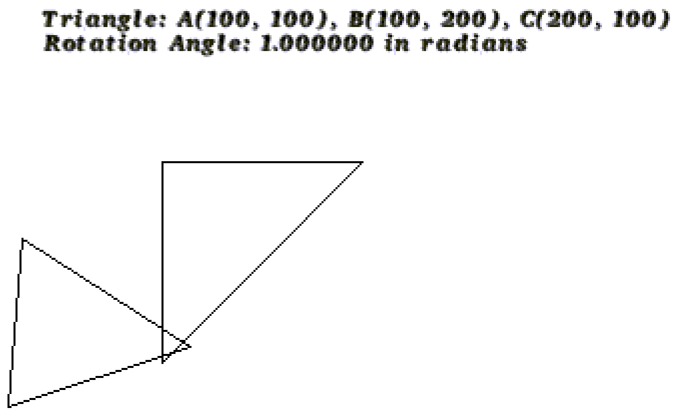
Points of Triangle: (100, 100), (100, 200), (200, 100).

Translation:
X: 100 Y: 50



```
Triangle: A(100, 100), B(100, 200), C(200, 100)
Translation x:100  y:50
```

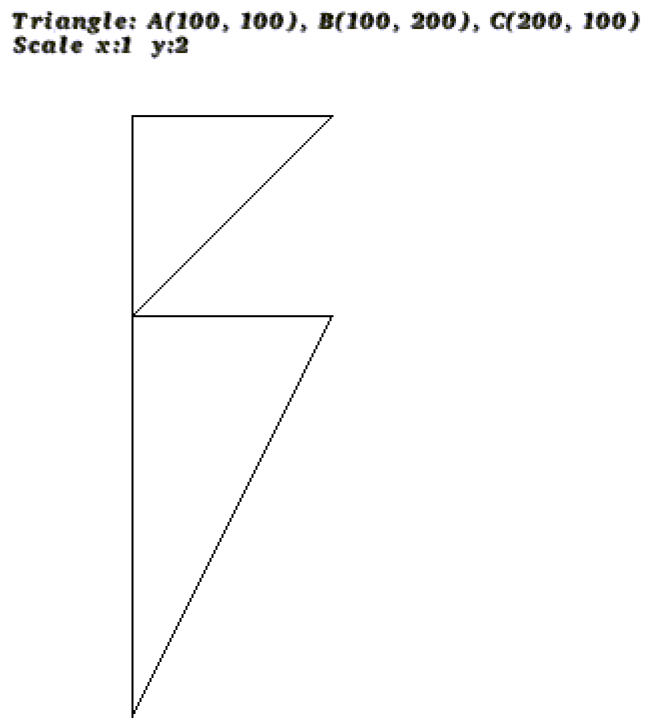
Rotation:
Angle: 1 radian



```
Triangle: A(100, 100), B(100, 200), C(200, 100)
Rotation Angle: 1.000000 in radians
```

Scaling:

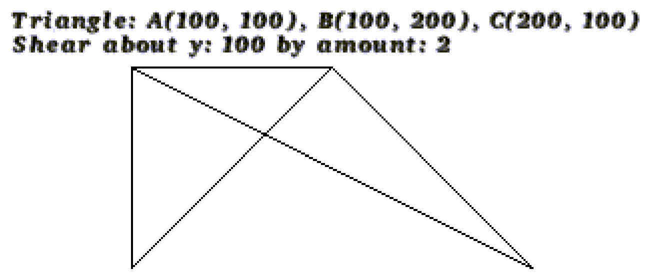
X: 1 Y: 2



```
Triangle: A(100, 100), B(100, 200), C(200, 100)
Scale x:1  y:2
```

Shear:

About y=100 and by an amount 2 in x direction



```
Triangle: A(100, 100), B(100, 200), C(200, 100)
Shear about y: 100 by amount: 2
```

PROGRAM 8

COHEN SUTHERLAND LINE CLIPPING

Description:

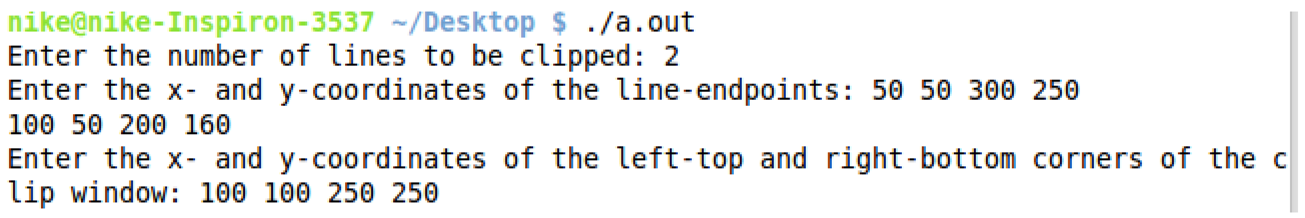
The Cohen–Sutherland algorithm is a computer graphics algorithm used for line clipping. The algorithm divides a two-dimensional space into 9 regions (or a three-dimensional space into 27 regions), and then efficiently determines the lines and portions of lines that are visible in the center region of interest (the viewport). The algorithm was developed in 1967 during flight simulator work by Danny Cohen and Ivan Sutherland.

Program:

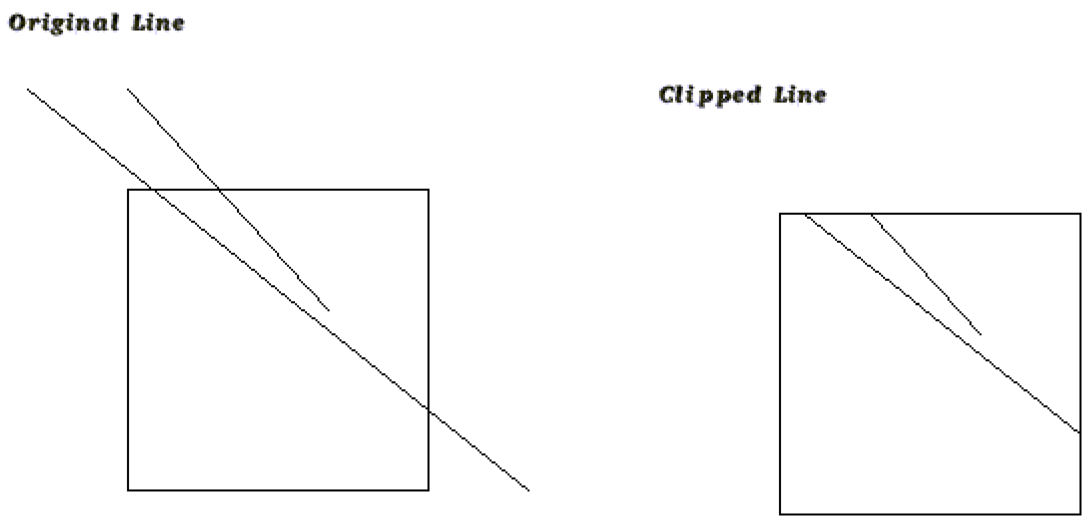
```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#define MAX 20

enum { TOP = 0x1, BOTTOM = 0x2, RIGHT = 0x4, LEFT = 0x8 };

enum { FALSE, TRUE };
typedef unsigned int outcode;

outcode compute_outcode(int x, int y,
                        int xmin, int ymin, int xmax, int ymax)
{
    outcode oc = 0;

    if (y > ymax)
            oc |= TOP;
    else if (y < ymin)
            oc |= BOTTOM;


    if (x > xmax)
            oc |= RIGHT;
    else if (x < xmin)
            oc |= LEFT;

    return oc;
}

void cohen_sutherland (double x1, double y1, double x2, double y2,
                        double xmin, double ymin, double xmax, double ymax)
{
    int accept;
    int done;
    outcode outcode1, outcode2;

    accept = FALSE;
    done = FALSE;

    outcode1 = compute_outcode (x1, y1, xmin, ymin, xmax, ymax);
    outcode2 = compute_outcode (x2, y2, xmin, ymin, xmax, ymax);
    do
    {
            if (outcode1 == 0 && outcode2 == 0)
            {
                accept = TRUE;
                done = TRUE;
            }
            else if (outcode1 & outcode2)
            {
                done = TRUE;
            }
            else
            {
                double x, y;
                int outcode_ex = outcode1 ? outcode1 : outcode2;
                if (outcode_ex & TOP)
                {
                        x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1);
                        y = ymax;
                }

                else if (outcode_ex & BOTTOM)
                {
                        x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1);
                        y = ymin;
                }
                else if (outcode_ex & RIGHT)
                {
                        y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1);
                        x = xmax;
                }
                else
                {
                        y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1);
                        x = xmin;
                }
                if (outcode_ex == outcode1)
                {
                        x1 = x;
                        y1 = y;
                        outcode1 = compute_outcode (x1, y1, xmin, ymin, xmax, ymax);
                }
                else
                {
                        x2 = x;
                        y2 = y;
                        outcode2 = compute_outcode (x2, y2, xmin, ymin, xmax, ymax);
                }
            }
    } while (done == FALSE);

    if (accept == TRUE)
            line (x1, y1, x2, y2);
}



int main()
{
    int n;
    int i, j;
    int ln[MAX][4];
    int clip[4];
    int gd = DETECT, gm;

    printf ("Enter the number of lines to be clipped: ");
    scanf ("%d", &n);

    printf ("Enter the x- and y-coordinates of the line-endpoints: ");
    for (i=0; i<n; i++)
                        for (j=0; j<4; j++)
                        scanf ("%d", &ln[i][j]);

    printf ("Enter the x- and y-coordinates of the left-top and right-");
    printf ("bottom corners of the clip window: ");
    for (i=0; i<4; i++)
                        scanf ("%d", &clip[i]);

    initgraph (&gd, &gm, NULL);
    setbkcolor(WHITE);
    setcolor(BLACK);
    setfontcolor(BLACK);
```

```
            printf("\n\tOriginal Line\n");
    rectangle (clip[0], clip[1], clip[2], clip[3]);
    for (i=0; i<n; i++)
                        line (ln[i][0], ln[i][1], ln[i][2], ln[i][3]);
            delay(5000);

    cleardevice();
    printf("\n\tClipped Line\n");
    rectangle (clip[0], clip[1], clip[2], clip[3]);
    for (i=0; i<n; i++)
    {
                        cohen_sutherland (ln[i][0], ln[i][1], ln[i][2], ln[i][3],
                clip[0], clip[1], clip[2], clip[3]);
    }
    delay(5000);

    closegraph();
}
```

Output:



```
nike@nike-Inspiron-3537 ~/Desktop $ ./a.out
Enter the number of lines to be clipped: 2
Enter the x- and y-coordinates of the line-endpoints: 50 50 300 250
100 50 200 160
Enter the x- and y-coordinates of the left-top and right-bottom corners of the c
lip window: 100 100 250 250
```



PROGRAM 9

LIANG BARSKY LINE CLIPPING

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:

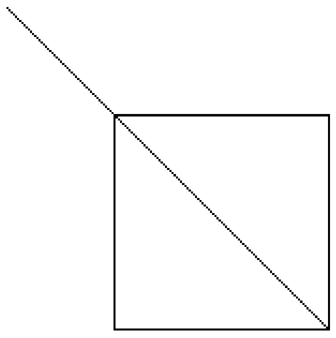
```
#include <stdio.h>
#include <stdlib.h>
#include<bits/stdc++.h>
using namespace std;
#include<graphics.h>

int main()
{
        int gd, gm ;
        int x1 , y1 , x2 , y2 ;
        int wxmin,wymin,wxmax, wymax ;
        float u1 = 0.0,u2 = 1.0 ;
        int p1 , q1 , p2 , q2 , p3 , q3 , p4 ,q4 ;
        float r1 , r2 , r3 , r4 ;
        int x11 , y11 , x22 , y22 ;
        printf("Enter the windows left xmin , top boundry ymin\n");
        scanf("%d%d",&wxmin,&wymin);
        printf("Enter the windows right xmax ,bottom boundry ymax\n");
        scanf("%d%d",&wxmax,&wymax);
        printf("Enter line x1 , y1 co-ordinate\n");
        scanf("%d%d",&x1,&y1);
        printf("Enter line x2 , y2 co-ordinate\n");
        scanf("%d%d",&x2,&y2);
        p1 = -(x2 - x1 ); q1 = x1 - wxmin ;
        p2 = ( x2 - x1 ) ; q2 = wxmax - x1 ;
        p3 = - ( y2 - y1 ) ; q3 = y1 - wymin ;
        p4 = ( y2 - y1 ) ; q4 = wymax - y1 ;

        if( ( ( p1 == 0.0 ) && ( q1 < 0.0 ) ) ||
        ( ( p2 == 0.0 ) && ( q2 < 0.0 ) ) ||
        ( ( p3 == 0.0 ) && ( q3 < 0.0 ) ) ||
( ( p4 == 0.0 ) && ( q4 < 0.0 ) ) )
{
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm,NULL);
        rectangle(wxmin,wymax,wxmax,wymin);
        line(x1,y1,x2,y2);
        line(x1,y1,x2,y2);
        delay(2000);
}
else{
        if( p1 != 0.0 )
        {
                r1 =(float) q1 /p1 ;
                if( p1 < 0 )
                        u1 = max(r1 , u1 );
                else
                        u2 = min(r1 , u2 );
        }
        if( p2 != 0.0 )
        {
                r2 = (float ) q2 /p2 ;
                if( p2 < 0 )
                        u1 = max(r2 , u1 );
                else
                        u2 = min(r2 , u2 );
        }
        if( p3 != 0.0 )
        {
                r3 = (float )q3 /p3 ;
                if( p3 < 0 )
                        u1 = max(r3 , u1 );
                else
                        u2 = min(r3 , u2 );
        }
        if( p4 != 0.0 )
        {
                r4 = (float )q4 /p4 ;
                if( p4 < 0 )
                        u1 = max(r4 , u1 );
                else
                        u2 = min(r4 , u2 );
        }

        if( u1 > u2 )
                printf("line rejected\n");
        else
```

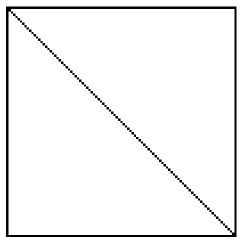
```
                {
                        x11 = x1 + u1 * ( x2 - x1 ) ;
                        y11 = y1 + u1 * ( y2 - y1 ) ;

                        x22 = x1 + u2 * ( x2 - x1 );
                        y22 = y1 + u2 * ( y2 - y1 );

                        detectgraph(&gd,&gm);
                        initgraph(&gd,&gm, NULL);
                        setbkcolor(WHITE);
                        setcolor(BLACK);
                        rectangle(wxmin,wymax,wxmax,wymin);
                        line(x1,y1,x2,y2);
                        line(x1,y1,x2,y2);
                        line(x11,y11,x22,y22);
                        delay(2000);
                }
        }
        return 0;
}
```

Output:

```
Line Points: (100, 100), (200, 200)
Window Left Corner: (150, 150)
Window Right Corner: (200, 200)
```

Before Clipping: **After Clipping:**





PROGRAM 10

BEZIER CURVE

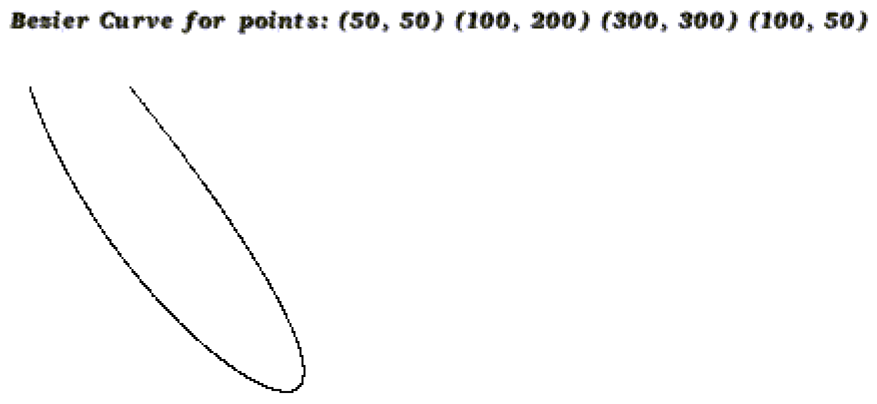
Description:

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. Bézier curves are also used in the time domain, particularly in animation, user interface design and smoothing cursor trajectory in eye gaze controlled interfaces.

Program:

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

int main(){
	int gd=DETECT,gm;
	int x[4],y[4],px,py,i;
	double t;

	// take input
	cout<<"Enter four control points for Bezier curve: ";
	for(i=0;i<4;i++)
		cin>>x[i]>>y[i];

	// graphics initialisation
	initgraph(&gd,&gm,NULL);
	setbkcolor(WHITE);
	setcolor(BLACK);
	setfontcolor(BLACK);

	// bezier curve
	printf("\n\tBezier Curve for points: (%d, %d) (%d, %d) (%d, %d) (%d, %d)\n", x[0], y[0], x[1], y[1],
x[2], y[2], x[3], y[3]);
	for(t=0.0;t<=1.0;t+=0.001){
		px=(1-t)*(1-t)*(1-t)*x[0]+3*t*(1-t)*(1-t)*x[1]+3*t*t*(1-t)*x[2]+t*t*t*x[3];
		py=(1-t)*(1-t)*(1-t)*y[0]+3*t*(1-t)*(1-t)*y[1]+3*t*t*(1-t)*y[2]+t*t*t*y[3];
		putpixel(px,py,getcolor());
		delay(2);
	}
	delay(5000);
	closegraph();
}
```

Output:



```
Besier Curve for points: (50, 50) (100, 200) (300, 300) (100, 50)
```

PROGRAM 11

CAR ANIMATION

Description:

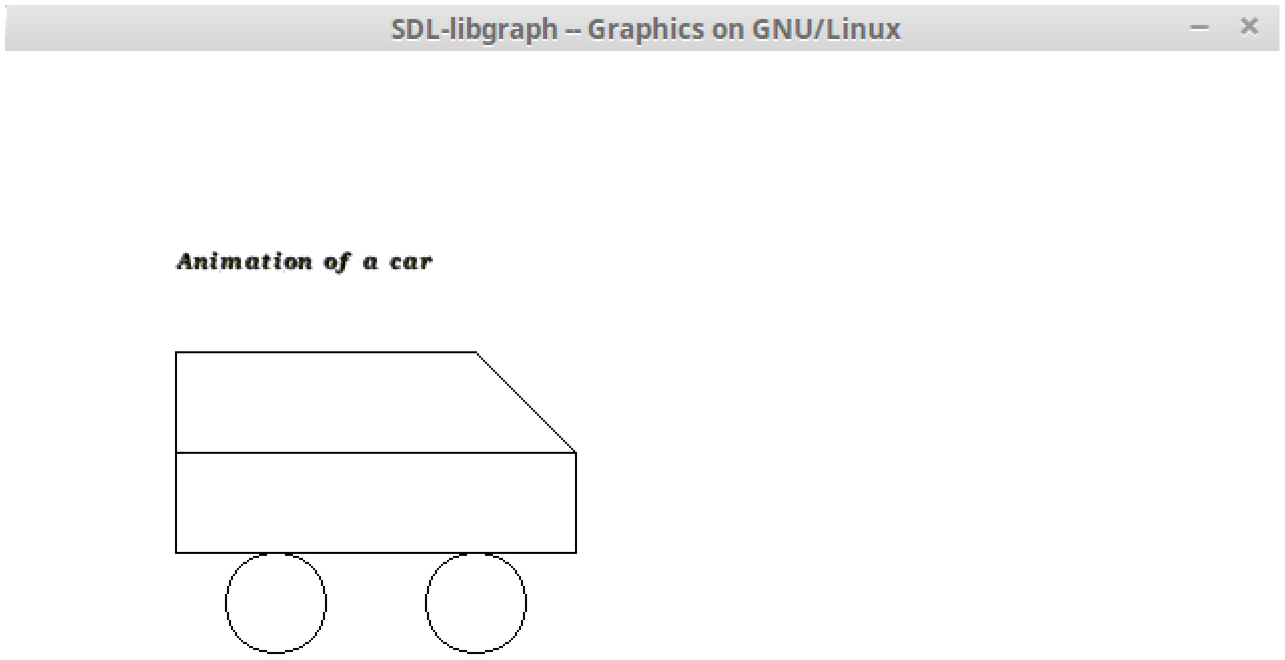
Computer animation, or CGI animation, is the process used for generating animated images. The more general term computer-generated imagery encompasses both static scenes and dynamic images, while computer animation only refers to the moving images. Modern computer animation usually uses 3D computer graphics, although 2D computer graphics are still used for stylistic, low bandwidth, and faster real-time renderings.

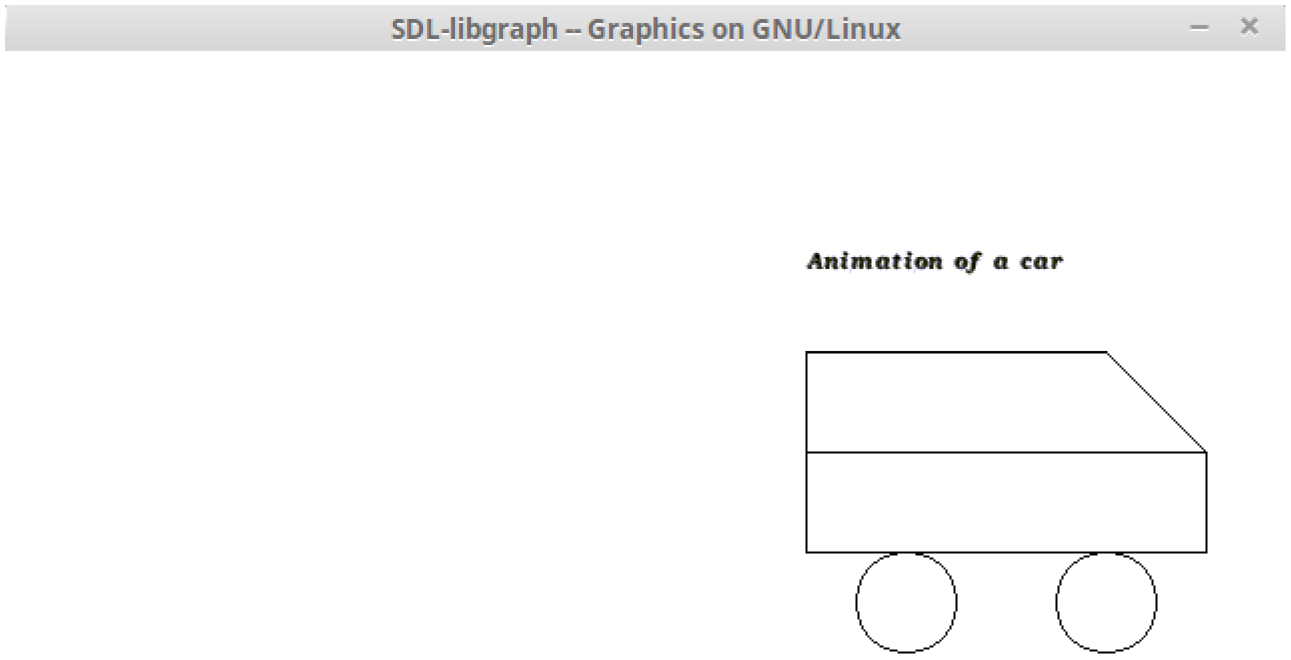
Program:

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

void drawCar(int xLeftLower,int yLeftLower){
  circle(xLeftLower+50, yLeftLower-25, 25);
  circle(xLeftLower+150, yLeftLower-25, 25);
  rectangle(xLeftLower, yLeftLower-100, xLeftLower+200, yLeftLower-50);
  line(xLeftLower, yLeftLower-100, xLeftLower, yLeftLower-150);
  line(xLeftLower, yLeftLower-150, xLeftLower+150, yLeftLower-150);
  line(xLeftLower+150, yLeftLower-150, xLeftLower+200, yLeftLower-100);
}

int main(int argc,char *argv[]){
  //graphics initialisation
  int gd = DETECT,gm;
  initgraph(&gd,&gm,NULL);
  setbkcolor(WHITE);
  setcolor(BLACK);
  setfontcolor(BLACK);

  //Draw the ellipse using Trignometric algo
  for(int x=50;x<=400;x++){
    cleardevice();
    outtextxy(x, 100, "Animation of a car\n");
    drawCar(x, 300);
    delay(25);
  }

  //delay so as to view the screen
  delay(5000);
        return 0;
```

}

Output:


SDL-libgraph -- Graphics on GNU/Linux
Animation of a car


SDL-libgraph -- Graphics on GNU/Linux
Animation of a car