SHRI VILEPARLE KELAVANI MANDAL'S

SHRI BHAGUBHAI MAFATLAL POLYTECHNIC

Computer Engineering Department

A

Laboratory Manual

Computer Graphics

(Course Code CGR198920)

Semester :IV

Vision /Mission Statements

Institute Vision

SBM Polytechnic aspires to be the lead institute in providing need based technical education.

Institute Mission

- **To provide state-of-the-art infrastructure and latest equipments for providing a stimulating learning environment.**
- **To prepare students to meet the dynamic needs of the industry by periodic reviewing and upgradation of curriculum through an interactive process with industry.**
- **To inculcate a spirit of excellence in terms of academic performance, research and innovation in faculty by providing appropriate support and incentive systems.**
- **To promote and support Co-Curricular, extra-curricular activities and industry interaction to make students socially sensitive and employable.**

Department Vision

Create a sustainable academic environment to produce highly competent computer professionals of the future

Department Mission

- **To expose students to latest tools and technologies in computing**
- **To foster the professional development of students by providing excellence in education.**
- **To Adapt rapid advancements in computing by engaging students in the lifelong learning.**
- **To inculcate sound ethical ,moral and social values amongst students for benefit of the society.**

Program Educational objectives

PEO1-Identify ,frame and solve computing problems by applying knowledge in Computer engineering

PEO2- Promote lifelong learning by integrating academic knowledge and practical applications

PEO3- Depict effective team work and practical skills for holistic development

Program Outcomes

PO 1. Basic knowledge: Apply subject knowledge of basic mathematics, sciences, and basic engineering to solve the Automobile engineering problems.

PO 2. Discipline knowledge: Solve Automobile engineering problems to bring optimal solutions within the constraints by applying Automobile engineering knowledge.

PO 3. Experiments and practice: Perform the experiments and interpret the data & results with a conclusion to give effective solutions to Automobile engineering problems.

PO 4. Engineering tools: Use tools and technologies with its understanding and limitations to get proper solutions of Automobile engineering problems.

PO 5. The engineer and society: Analyze societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to practice in a field of Automobile Engineering.

PO 6. Environment and sustainability: Solve social and environmental problems for sustainable development by applying Automobile engineering solutions.

PO 7. Ethics: Apply ethical principles for commitment to professional ethics, responsibilities, and norms of the practice also in the field of Automobile engineering.

PO 8. Individual and teamwork: Function effectively as a team leader or member of multidisciplinary groups.

PO 9. Communication: Communicate & present effectively in oral and written form.

PO 10. Life-long learning: Adapt the importance of lifelong learning and retain the capacity to do the same in Automobile Engineering & Allied industry.

Program Specific Outcomes

PSO1: Demonstrate the fundamental knowledge in the areas of Operating system, Web Technology, Microprocessor based system and IOT by applying programming skills and developing applications

PSO2: Administer and manage Open source, Networking, Security and Database domains to enhance student growth

Course Outcomes

At the end of the semester student will be able to: -

1 Conceptualize various display devices with its usage .
2 Implement the algorithms namely line drawing, circle drawing, ellipse drawing and area filling
3 Transform 2D and 3D objects.
4 Implement Line / Polygon Clipping algorithms
5 Appraise the advanced graphics topics.

CO-PO Mapping

Course Outcomes (CO)	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PSO 1	PSO 2
CO 1	2							1	
CO 2	3	1	2					3	
CO 3	2	1	2					2	
CO 4	2	2	2					2	
CO 5	2							1	
	2.2	1.33	2					1.8	

Index

Experiment 1 : Program for Pixel Drawing

Aim : To write a program for pixel drawing

Theory :

C graphics using graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of graphics.h in Turbo C compiler we can make graphics programs, animations, projects, and games. We can draw circles, lines, rectangles, bars and many other geometrical figures. You can change their colors using the available functions and fill them.

Basic Structure of a C-graphics program:

```
#include<stdio.h>
#include<graphics.h>//must be included for every graphics program
#include<conio.h>
#include<dos.h> //for including delay function.
void main()
{
        int gd=DETECT,gm; //gd=detects best available graphics driver, gm =graphics mode.
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");// for initializing graph mode
        // above 2 steps are must for every graphics program.
        //declaration of any variables must be done before calling initgraph() function.
        // next write code for producing requiring design or drawing object
        line(100,100,200,200);//draws a line segment.
        getch();
}
```

putpixel function in c

putpixel function plots a pixel at location (x, y) of specified color.
Declaration: void putpixel(int x, int y, int color);
For example,if we want to draw a GREEN color pixel at (35, 45) then we will write putpixel(35, 35, GREEN); in our c program, putpixel function can be used to draw circles, lines and ellipses using various algorithms.

```
#include<graphics.h>
#include<conio.h>
void main()
{
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "C:\\TC\\BGI");
  putpixel(25, 25, RED);
  getch();
  closegraph();
```

}

Output of this program will be a RED pixel on screen at (25, 25) .

Code

Sr.no	Name	Uses	Syntax
1.	Circle	Circle function is used to draw the circle on the screen	circle(x,y,radius);
2.	Closegraph	closegraph function shut down the graphic system	closegraph();
3.	Cleardevice	cleardevice function is used to clear the contents or graphic images on the screen in graphics mode	cleardevice();
4.	Outtextxy	outtextxy function is used to print the text on the screen in graphics mode.	outtext(x,y,text);
5.	Floodfill	floodfill function is used to fill the color in the object, object may be circle, rectangle or any other closed image.	floodfill(x,y,boundary color);
6.	Setcolor	setcolor is to set color of the objects which is to be drawn after this setcolor line.	setcolor(COLOR);
7.	Getmaxx	getmaxxreturns the maximum x co-ordinate on the screen	getmaxx();
8.	Line	Line function is used to draw the line on the screen.	line(x1,y1,x2,y2);
9.	Rectange	Rectangle function is used to draw the rectangle on the screen. X1,y1 are the lower left co-ordinates of the rectangle and the x2,y2 are the upper right co-ordinates of the rectangle.	rectangle(x1,,y1,x2,y2);
10.	Ellipse	ellipse function is used to draw the ellipse on the screen.	ellipse(x, y, starting angle, ending angle, xradius, yradius);
11.	Putpixel	Putpixel function is to draw the pixel on the screen. Pixel is small dot on the screen.	putpixel(x co-orinate, y co-ordinate,COLOR);
12.	Getcolor	getcolor returns the current drawing color.	getcolor();
13.	Getpixel	getpixel gets the color of a specified pixel.	getpixel(x,y);
14.	Detectgraph	Determines graphics drivers and mode to use by checking the hardware.	Void far detectgraph(int far*graphicsdriver,int far*graphmode);

15.	initgraph	To start the graphics system, you must first call initgraph. Initgraph initializes the graphics system by loading a graphics driver from disk (or validating a registered driver)then putting the system into graphics mode	Void far initgraph(iny far* graphicsdriver,int far*graphmode,char far*pathodriver);

Output

Conclusion: We learnt about the different function used in computer graphics code.

Sample Questions :

1 What is the syntax of putpixel function

2 Identify the functions for drawing circle,rectangle,arc

Experiment 2 : Program for Line drawing by DDA

Aim: To Write Program for Line drawing using DDA algorithm

Theory :

DDA(Digital differential analyzer)algorithm is an incremental scan conversion method. Here we perform calculations at each step using the results from the preceding step. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along the other coordinate. The unit steps are always along the coordinate of greatest change, e.g. if dx = 10 and dy = 5, then we would take unit steps along x and compute the steps along y**.**

Algorithm:

1. Start.
2. Declare variables x,y,x1,y1,x2,y2,k,dx,dy,s,xi,yi and also declare
gdriver=DETECT, mode.
3. Initialize the graphic mode with the path location in TurboC3 folder.
4. Input the two line end-points and store the left end-points in (x1,y1).
5. Load (x1, y1) into the frame buffer; that is, plot the first point. put x=x1,y=y1.
6. Calculate dx=x2-x1 and dy=y2-y1.
7. If abs (dx) > abs (dy), do s=abs(dx).
8. Otherwise s= abs(dy).
9. Then xi=dx/s and yi=dy/s.
10. Start from k=0 and continuing till k<s,the points will be
i. x=x+xi.
ii. Y=y+yi.
11. Plot pixels using putpixel at points (x,y) in specified colour.
12. Close Graph and stop

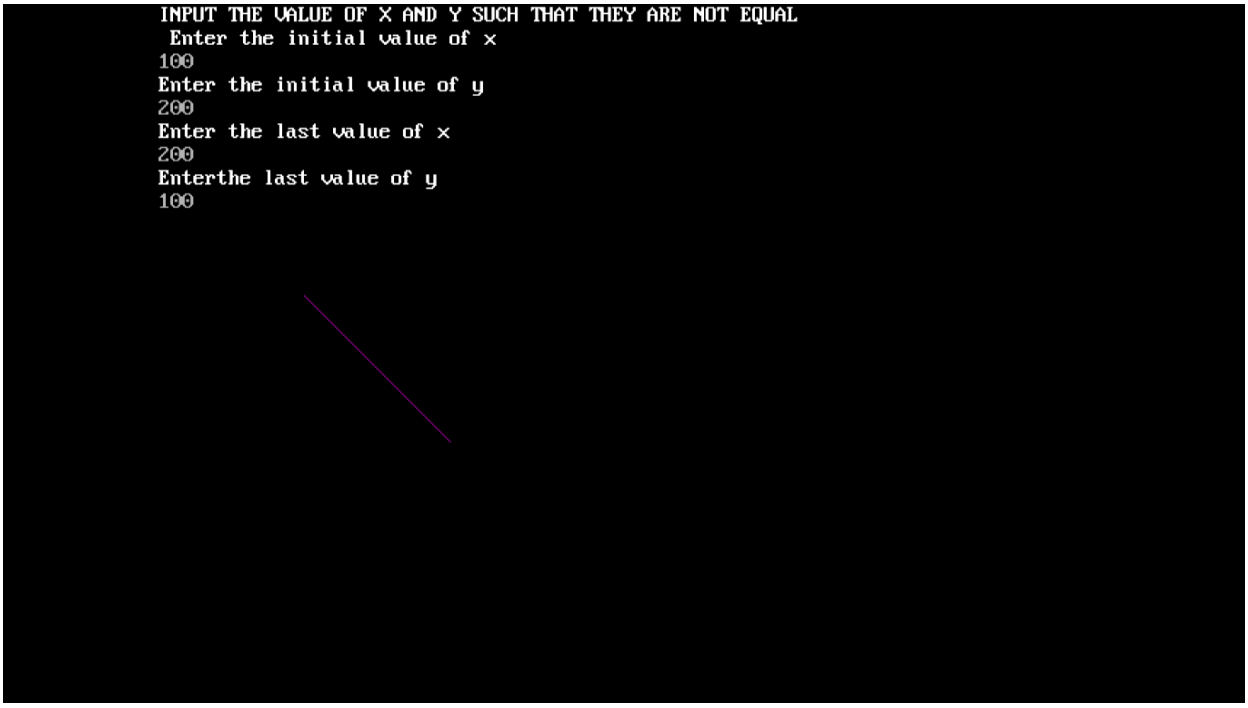
Code

```
#include <graphics.h>

#include <iostream.h>

#include <stdio.h>

#include <conio.h>

#include<math.h>

#include<dos.h>

int main()

{


  int gdriver = DETECT, gmode;
```

```
  float x,y,x1,y1,x2,y2,L,dx,dy,dx1,dy1;
  int i=0;
  initgraph(&gdriver, &gmode, "C:\\TURBOC3\\bgi");
 setcolor(getmaxcolor());
  printf("INPUT THE VALUE OF X AND Y SUCH THAT THEY ARE NOT EQUAL\n ");
  printf("Enter the initial value of x\n");
  scanf("%f",&x1);
  printf("Enter the initial value of y \n");
  scanf("%f",&y1);
  printf("Enter the last value of x \n");
  scanf("%f",&x2);
  printf("Enterthe last value of y \n");
  scanf("%f",&y2);
  dx=abs(x2-x1);
  dy=abs(y2-y1);
  if(dx>=dy)
  { L=dx; }
  else
  {  L=dy; }
  dx1=dx/L;
  dy1=dy/L;
  x=x1+0.5;
  y=y1+0.5;
  printf("i \t x \t \t  y \t \n");
  while(i<=L)
  {
    putpixel(x,y,5);
    x=x+dx1;
    y=y+dy1;
```

```
    printf("%d \t %f \t %f \n",i,x,y);
    i++;
  }
  /* clean up */
  getch();
  closegraph();
  return 0;
}
```

Output



Conclusion: From this experiment we learnt about the line drawing algorithm i.e. DDA line algorithm and its derivation, also implemented the algorithm using cpp code.

Questions :

1) What is DDA Algorithm?
2) What are the merits and demerits of DDA algorithm

Experiment 3 : Program for Line drawing by Bresenham's

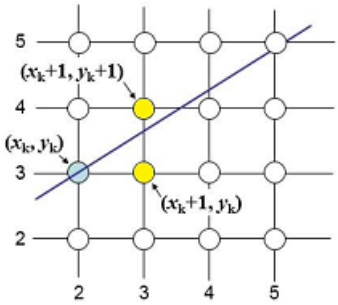
Aim: To Write a Program for Line drawing using Bresenham's Line Drawing algorithm

Theory:

Bresenham's line algorithm is an algorithm which determines which order to form a close approximation to a straight line between two given points.

Basic Concept:

Move across the x axis in unit intervals and at each step choose between two different y coordinates



For example, from position (2, 3) we have to choose between (3, 3) and (3, 4). We would like the point that is closer to the original line

So we have to take decision to choose next point. So next pixels are selected based on the value of decision parameter p. The equations are given in below algorithm.

BRESENHAM'S LINE DRAWING ALGORITHM

1. Input the two line end-points, storing the left end-point in (x0, y0)
2. Plot the point (x0, y0)
3. Calculate the constants Δx, Δy, 2Δy, and (2Δy - 2Δx) and get the first value for the decision parameter as:

$$p_0 = 2\Delta y - \Delta x$$

4. At each xk along the line, starting at k = 0, perform the following test. If pk < 0, the next point to plot is (xk+1, yk) and:

$$p_{k+1} = p_k + 2\Delta y$$

Otherwise, the next point to plot is (xk+1, yk+1) and:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

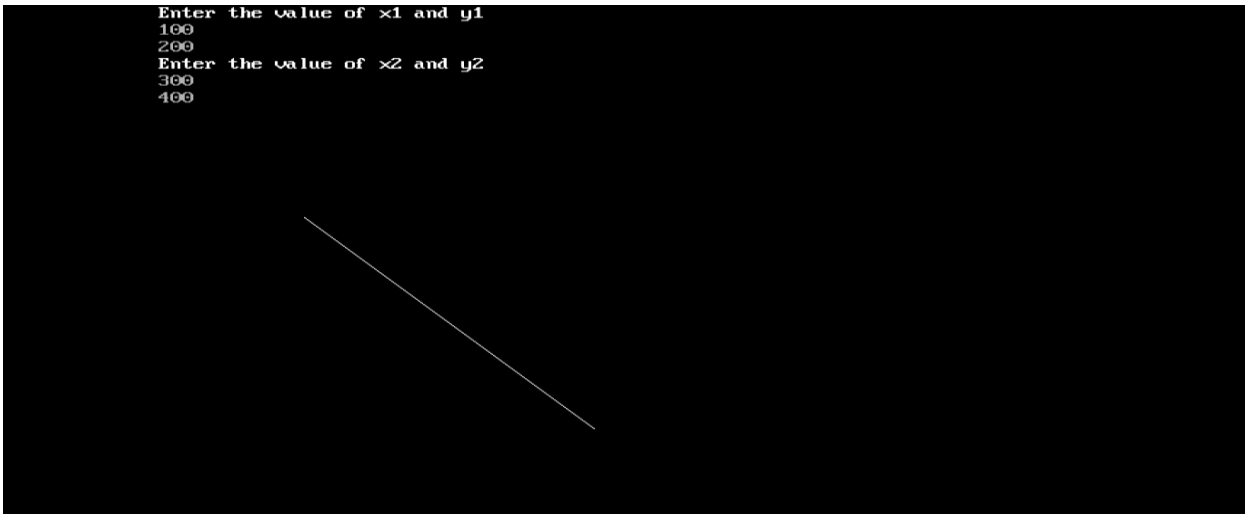
5. Repeat step 4 (Δx – 1) times

Code:

```
#include< conio.h>
#include<stdio.h>
#include<GRAPHICS.H>

int main()

{
   int gdriver=DETECT,gmode;
   int i,e,x,y,x1,y1,x2,y2,Dx,Dy;
   initgraph(&gdriver,&gmode,"C:\\TURBOC3\\bgi");
   printf("Enter the value of x1 and y1");
   scanf("%d %d",&x1,&y1);
```

```
   printf("Enter the value of x2 and y2");
  scanf("%d %d",&x2,&y2);
  Dx=abs(x2-x1);
  Dy=abs(y2-y1);
  x=x1;
  y=y1;
  e=2*Dy-Dx;
  i=1;
 printf("i \t x \t  y \t  e \n");
 do
 {
 putpixel(x,y,15);

  while(e>=0)
  {
   y=y+1;
   e=e-2*Dx;
  }
  x=x+1;
  e=e+2*Dy;
 // printf("%d \t %d \t %d \t %d \n ",i,x,y,e);
  i++;
  }
  while(i<=Dx);
  getch();
  closegraph();
  return 0;}
```

Output:



Conclusion: We learnt about the another line drawing algorithm i.e. bresenham's line drawing algorithm and studied its derivation along with implementing its code in c.

Questions :

What is Need for Bresenhams algorithm?
What is Decision parameter and what is its need?

Experiment 4: Program for Circle Drawing by DDA

Aim : To write a Program for Circle Drawing by DDA

Theory :

In this algorithm the calculations have been done assuming the center of the circle is at (0, 0) and we have a procedure (circlePoints(CX, CY, X, Y)) that takes advantage of the symmetries, and turns on 8 pixels once we specify the center of the circle (CX, CY) and the pixel to start with relative to the center.

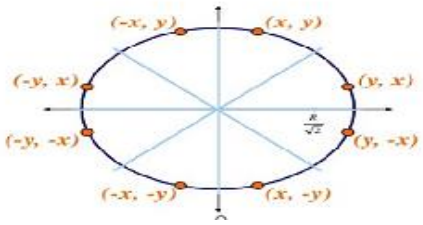


FIGURE 1: 8 POINT SYMMETRY OF CIRCLE

DDA CIRCLE GENERATION ALGORITHM

```
X = 0
Y = R
D = 1 - R
circlePoints(CX, CY, X, Y)
While (X < Y)
        If D < 0
        D = D + 2 * X + 3
        Else
        Y = Y - 1
        D = D + 2 * (X - Y) + 5
        EndIf
        X = X + 1
        circlePoints(CX, CY, X, Y)
EndWhile
```

Conclusion: From this experiment we learnt about DDA circle drawing algorithm along with its derivation, and implementing using the cpp code.

Questions :

1 What is 8 -way symmetry of a circle?

2 How DDA circle takes the advantage of 8 way symmetry to draw a circle?

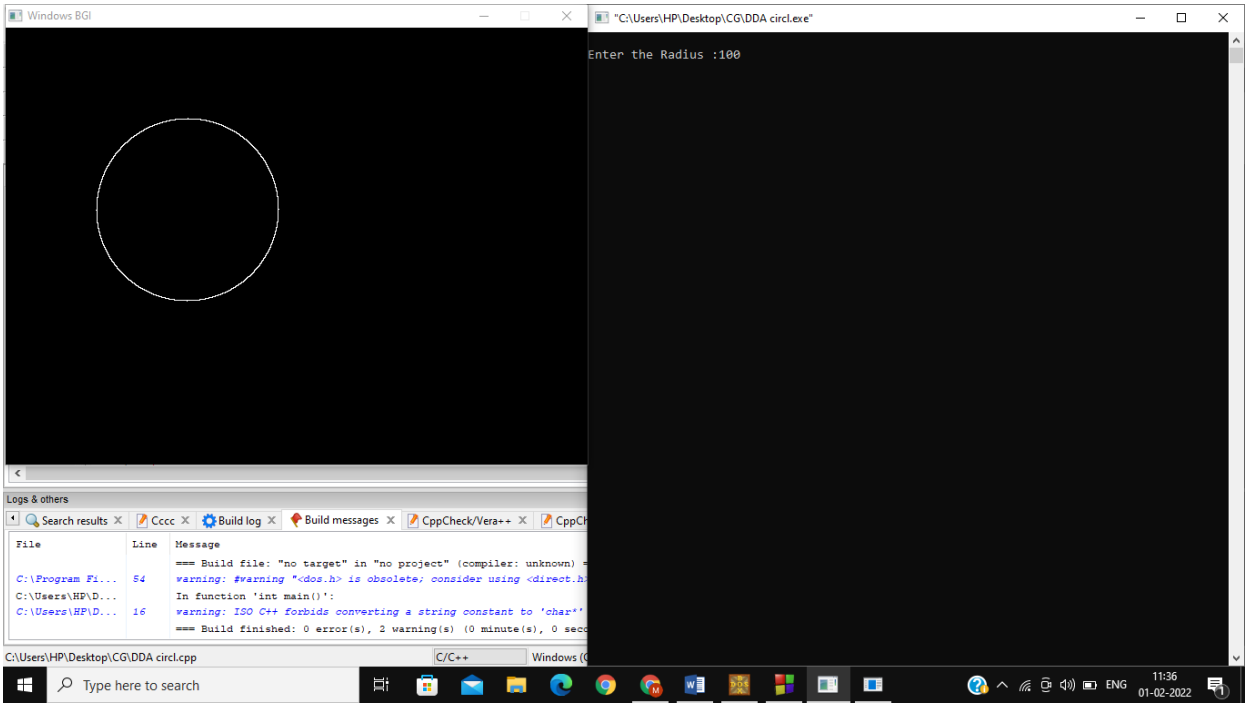
Code:

```
#include<iostream>

#include<conio.h>
```

```
#include<dos.h>
#include<math.h>
#include<graphics.h>
using namespace std;
int main()
{

  int i,val,r,gd,gm;
 float x1,y1,x2,y2,startx,starty,epsilon;
  cout<<"\nEnter the Radius :";
  cin>>r;
  detectgraph(&gd,&gm);
  initgraph(&gd,&gm,"");
   x1=0;
   y1=r;
   startx=x1;
   starty=y1;
   i=0;
   do
   {
      val=pow(2,i);
      i++;
   }while(val<r);
   epsilon=1/pow(2,i-1);
   do
   {
      x2=x1+y1*epsilon;
      y2=y1-epsilon*x2;
      putpixel(200+x2,200+y2,15);
```

```
        x1=x2;
        y1=y2;
    }while((y1-starty)<epsilon||(startx-x1)>epsilon);
    getch();
    closegraph();
}
```

Output:



Experiment 5 : Program for Circle Drawing by Bresenham's

Aim: To write a program for circle drawing by Bresenhams circle drawing algorithm

Theory:

Bresenham's Circle Drawing Algorithm is a circle drawing algorithm that selects the nearest pixel position to complete the arc. The unique part of this algorithm is that is uses only integer arithmetic which makes it, significantly, faster than other algorithms using floating point arithmetic in classical processors.

ALGORITHM

- Set initial values of (xc, yc) and (x, y)
- Set decision parameter d to d = 3 – (2 * r).
- call drawCircle(int xc, int yc, int x, int y) function.
- Repeat steps 5 to 8 until x < = y
- Increment value of x.
- If d < 0, set d = d + (4*x) + 6
- Else, set d = d + 4 * (x – y) + 10 and decrement y by 1.
- call drawCircle(int xc, int yc, int x, int y) function

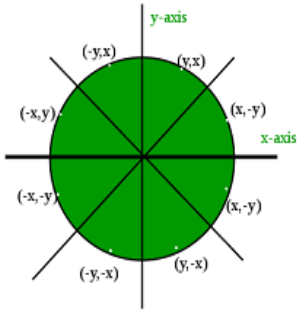


FIGURE 1: For a pixel (x, y) find all possible pixels in 8 octants

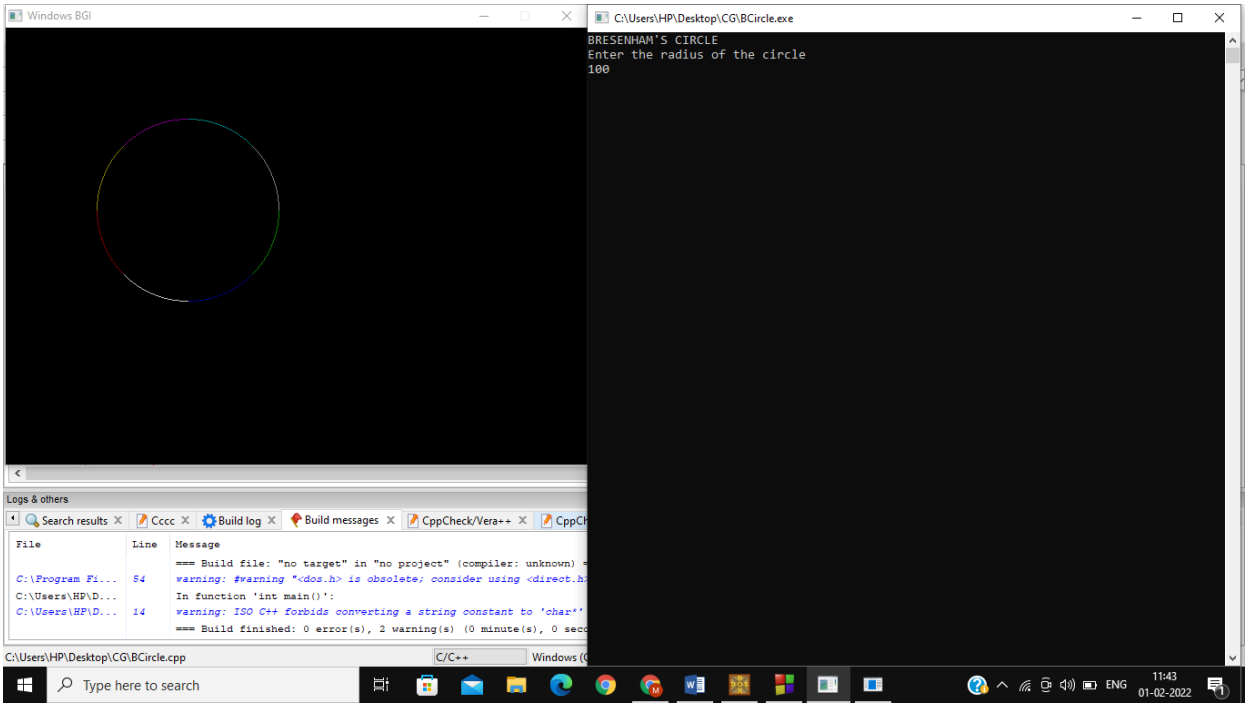
Code:

```
#include<iostream>

#include<math.h>

#include<conio.h>

#include<dos.h>

#include<graphics.h>

using namespace std;
```

```
int main()
{
 int gdriver=DETECT,gmode;
 int r,d,x,y;
 detectgraph(&gdriver,&gmode);
 initgraph(&gdriver,&gmode,"");
 cout<<"BRESENHAM'S CIRCLE "<<endl;
 cout<<"Enter the radius of the circle"<<endl;
 cin>>r;
  x=0;
  y=r;
 d=3-2*r;
 do
 {
   putpixel(x+200,y+200,1);
   putpixel(y+200,x+200,2);
   putpixel(x+200,-y+200,3);
   putpixel(-y+200,x+200,4);
   putpixel(-x+200,-y+200,5);
   putpixel(-y+200,-x+200,6);
   putpixel(-x+200,y+200,7);
   putpixel(y+200,-x+200,8);
   if(d<0)
   {
     d=d+4*x+6;
   }
   else
   {
```

```
    d=d+4*(x-y)+10;
    y=y-1;
  }
  x=x+1;
 }while(x<y);
 getch();
 return 0;
 closegraph();}
```

Output:



Conclusion: From this algorithm we learnt about the Bresenhams circle drawing algorithm implementing its code and studied its derivation.

Questions :

1) How decision parameter calculated in Bresenhams circle drawing algorithm?

2) What are the advantages and disadvantages of Bresenhams circle drawing algorithm?

Experiment 6 : Program for Circle Drawing by midpoint

Aim: Write a program for circle drawing by midpoint circle generation algorithm

Theory:

Given the centre point and radius of circle,Mid-Point Circle Drawing Algorithm attempts to generate the points of one octant.The points for other octants are generated using the eight-way symmetry property. The mid-point circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle. We use the mid-point algorithm to calculate all the perimeter points of the circle in the first octant and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about it's centre.

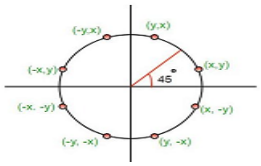


FIGURE: SYMMETRIC POINTS ABOUT CENTER OF A CIRCLE

The algorithm is very similar to the Mid-Point Line Generation Algorithm. Here, only the boundary condition is different. For any given pixel (x, y), the next pixel to be plotted is either (x, y+1) or (x-1, y+1). This can be decided by following the steps below.

- Find the mid-point p of the two possible pixels i.e (x-0.5, y+1)
- If p lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it's outside we plot the pixel (x-1, y+1)
- Boundary Condition : Whether the mid-point lies inside or outside the circle can be decided by using the formula:-
    - Given a circle centered at (0,0) and radius r and a point p(x,y)
    - $F(p) = x^2 + y^2 – r^2$
    - if $F(p)<0$, the point is inside the circle
    - $F(p)=0$, the point is on the perimeter
    - $F(p)>0$, the point is outside the circle

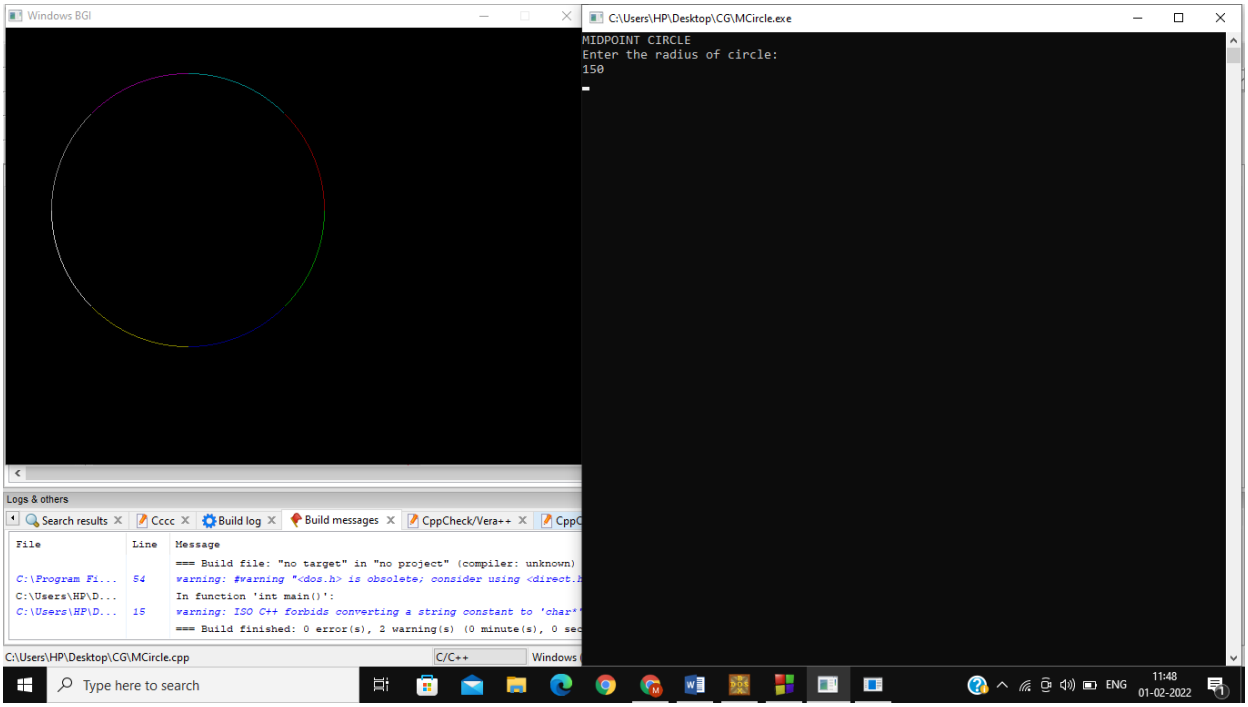
Code:

```
#include<iostream>

#include<math.h>

#include<dos.h>

#include<conio.h>

#include<graphics.h>

using namespace std;
```

```
int main()
{
 int gdriver=DETECT,gmode;
 int r;
 float d,x,y;
 detectgraph(&gdriver,&gmode);
 initgraph(&gdriver,&gmode,"");
 cout<<"MIDPOINT CIRCLE"<<endl;
 cout<<"Enter the radius of circle:"<<endl;
 cin>>r;
 x=0;
 y=r;
 d=1.25-r;
 do
 {
  putpixel(200+x,200+y,1);
  putpixel(200+y,200+x,2);
  putpixel(200+x,200-y,3);
  putpixel(200+y,200-x,4);
  putpixel(200-x,200-y,5);
  putpixel(200-x,200+y,6);
  putpixel(200-y,200+x,7);
  putpixel(200-y,200-x,8);
  if(d<0)
  {
   x=x+1;
   y=y;
   d=d+2*x+1;   }
```

```
  else   {
    x=x+1;
    y=y-1;
    d=d+2*(x-y)+1;
  }
 }while(x<y);
 getch();
 return 0;
 closegraph();}
```

Output:



Conclusion: From this experiment we learnt about midpoint circle drawing algorithm along with its derivation and implemented it using cpp code.

Questions:

1)What is boundary condition in midpoint circle generation?
2)Write the formula for boundary condition?

Experiment 7 : Program for Ellipse Drawing (midpoint)

Aim : To Implement Midpoint Ellipse Drawing algorithm

Theory :

Mid-point Ellipse algorithm is used to draw an ellipse in computer graphics. Midpoint ellipse algorithm plots(finds) points of an ellipse on the first quadrant by dividing the quadrant into two regions.Each point(x, y) is then projected into other three quadrants (-x, y), (x, -y), (-x, -y) i.e. it uses 4-way symmetry.

Function of ellipse:

$f_{ellipse}(x, y)=r_y^2x^2+r_x^2y^2-r_x^2r_y^2$

$f_{ellipse}(x, y)<0$ then (x, y) is inside the ellipse.

$f_{ellipse}(x, y)>0$ then (x, y) is outside the ellipse.

$f_{ellipse}(x, y)=0$ then (x, y) is on the ellipse.

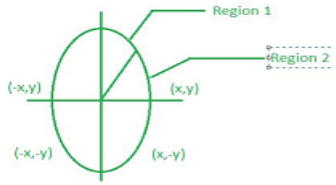


FIGURE : DIVISION OF FIRST QUADRANT INTO REGIONS

Mid-Point Ellipse Algorithm :

1. Take input radius along x axis and y axis and obtain center of ellipse.
2. Initially, we assume ellipse to be centered at origin and the first point as : $(x, y_0)= (0, r_y)$.
3. Obtain the initial decision parameter for region 1 as: $p1_0=r_y^2+1/4r_x^2-r_x\,^2r_y$
4. For every x_k position in region 1 :
   If $p1_k<0$ then the next point along the is (x_{k+1}, y_k) and $p1_{k+1}=p1_k+2r_y^2x_{k+1}+r_y^2$
   Else, the next point is (x_{k+1}, y_{k-1})
   And $p1_{k+1}=p1_k+2r_y^2x_{k+1} – 2r_x^2y_{k+1}+r_y^2$
5. Obtain the initial value in region 2 using the last point (x_0, y_0) of region 1 as:
   $p2_0=r_y^2(x_0+1/2)^2+r_x^2 (y_0-1)^2-r_x^2r_y^2$
6. At each y_k in region 2 starting at k =0 perform the following task.
   If $p2_k>0$ the next point is (x_k, y_{k-1}) and $p2_{k+1}=p2_k-2r_x^2y_{k+1}+r_x^2$
7. Else, the next point is (x_{k+1}, y_{k-1}) and $p2_{k+1}=p2_k+2r_y^2x_{k+1} -2r_x^2y_{k+1}+r_x^2$
8. Now obtain the symmetric points in the three quadrants and plot the coordinate value as: x=x+xc, y=y+yc
9. Repeat the steps for region 1 until $2r_y^2x>=2r_x^2y$

Code:

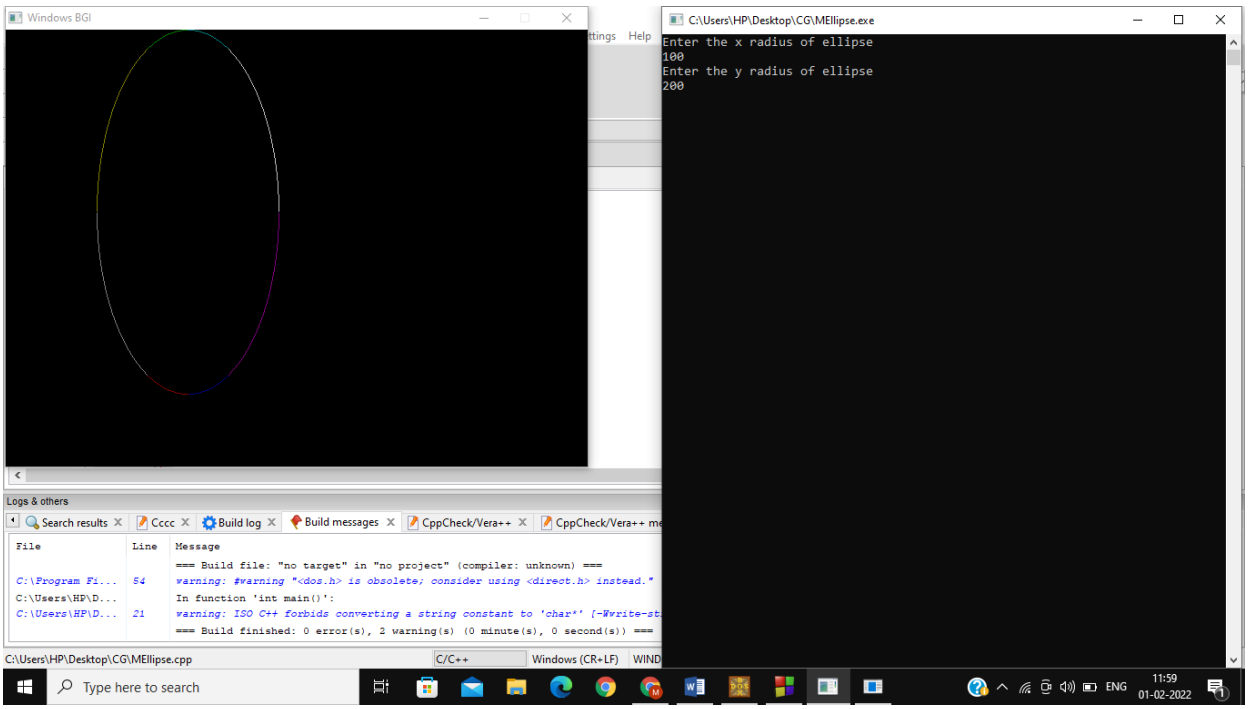
```
#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>
```

```
#include<dos.h>
using namespace std;
int main()
{
 int gdriver=DETECT,gmode;
 long d1,d2;
 int i,y,x;
 long rx,ry,rxsq,rysq,tworxsq,tworysq,dx,dy;
 cout<<"Enter the x radius of ellipse"<<endl;
 cin>>rx;
 cout<<"Enter the y radius of ellipse"<<endl;
 cin>>ry;
 detectgraph(&gdriver,&gmode);
 initgraph(&gdriver,&gmode," ");
 rxsq=rx*rx;
 rysq=ry*ry;
 tworxsq=2*rxsq;
 tworysq=2*rysq;
 x=0;
 y=ry;
 d1=rysq-rxsq*ry+(0.25*rxsq);
 dx=tworysq*x;
 dy=tworxsq*y;
 do
 {
  putpixel(200+x,200+y,1);
   putpixel(200-x,200-y,2);
    putpixel(200+x,200-y,3);
     putpixel(200-x,200+y,4);
```

```
    if(d1<0)
    {
     x=x+1;
     y=y;
     dx=dx+tworysq;
     d1=d1+dx+rysq;
    }
    else   {
     x=x+1;
     y=y-1;
     dx=dx+tworysq;
     dy=dy-tworxsq;
     d1=d1+dx-dy+rysq;  }
    delay(10);
 }while(dx<dy);
 d2=rysq*(x+0.5)*(x+0.5)+rxsq*(y-1)*(y-1)-rxsq*rysq;
 do  {
  putpixel(200+x,200+y,5);
   putpixel(200-x,200-y,6);
    putpixel(200+x,200-y,7);
     putpixel(200-x,200+y,8);
     if(d2>0)    {
      x=x;
      y=y-1;
      dy=dy-tworxsq;
      d2=d2-dy+rxsq;
     }
     else
```

```
    {
     x=x+1;
     y=y-1;
     dy=dy-tworxsq;
     dx=dx+tworysq;
     d2=d2+dx-dy+rxsq;
    }
 }while(y>0);
 getch();
 closegraph();}
```

Output:



Conclusion: We studied about ellipse drawing algorithm along with its derivation and implemented it using cpp code.

Questions :

1)What is major and minor axis in Ellipse?
2)What is the function of ellipse?
3)In Mid-point ellipse drawing why do we divide qudrants to regions?

Experiment 8 : Program for Transformation (2D)

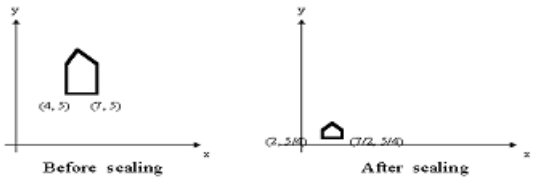
Aim: To Implement basic 2D transformations such as translation, Scaling, Rotation, shearing and reflection for a given 2D object.

Theory:

The 2D transformations are:
1. Translation
2. Scaling
3. Rotation
4. Reflection
5. Shear

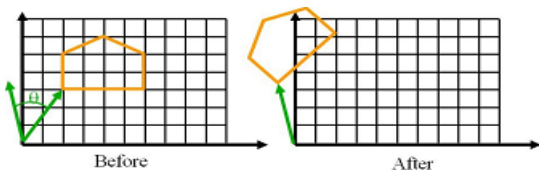
1. Translation: Translation is defined as moving the object from one position to another position along straight line path.

2. Scaling: scaling refers to changing the size of the object either by increasing or decreasing. We will increase or decrease the size of the object based on scaling factors along x and y-axis.

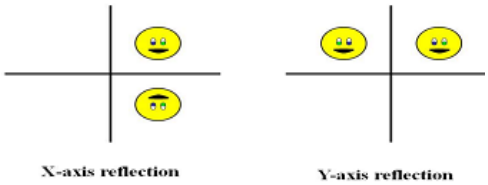


3. Rotation: A rotation repositions all points in an object along a circular path in the plane centered at the pivot point. We rotate an object by an angle theta. New coordinates after rotation depend on both x and y

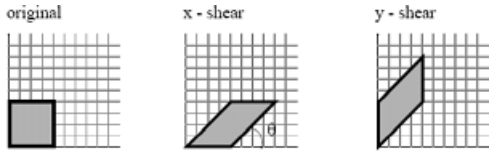
- $x' = x\cos\theta - y\sin\theta$
- $y' = x\sin\theta + y\cos\theta$



4. Reflection: Reflection is nothing but producing mirror image of an object. Reflection can be done just by rotating the object about given axis of reflection with an angle of 180 degrees.



5.Shear: Shear transformation is of 2 types:.
X-shear: changing x-coordinate value and keeping y constant
Y-shear: changing y coordinates value and keeping x constant



Code:

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int main()
{

   int gdriver=DETECT,gmode;
   initgraph(&gdriver,&gmode,"C:\\TURBOC3\\BGI");
   int i,j,k,x1,y1,x2,y2,choice;
   int a[3][3],t[3][3],r1[3][3],s[3][3],r2[3][3],r3[3][3],ro[3][3];
   cout<<"Enter the value of original matrix: "<<endl;
   for(i=0;i<3;i++)
   {
    for(j=0;j<3;j++)
    {
     cin>>a[i][j];
    }
   }
```

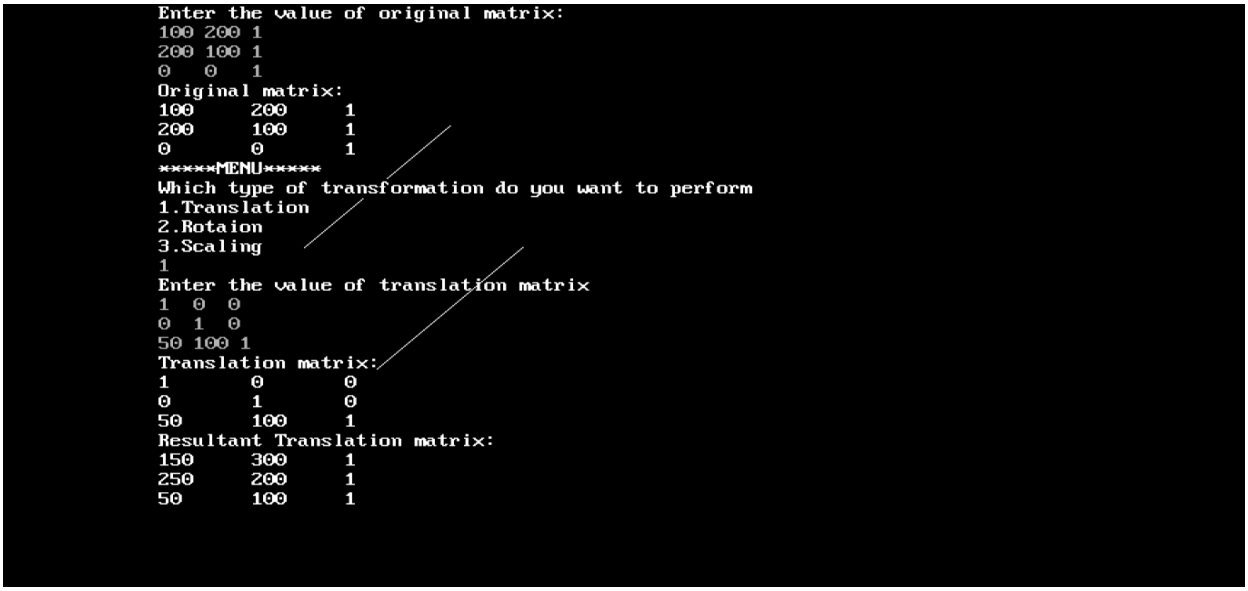
```
x1=a[0][0];
y1=a[0][1];
x2=a[1][0];
y2=a[1][1];
 cout<<"Original matrix: "<<endl;
for(i=0;i<3;i++)    {
 for(j=0;j<3;j++)     {
  cout<<a[i][j]<<"\t";     }
 cout<<endl;    }
line(x1,y1,x2,y2);
 cout<<"*****MENU*****"<<endl;
 cout<<"Which type of transformation do you want to perform "<<endl;
 cout<<"1.Translation "<<endl;
 cout<<"2.Rotaion "<<endl;
 cout<<"3.Scaling"<<endl;
 cin>>choice;
switch(choice)
{
  case 1:{
                                        cout<<"Enter the value of translation matrix"<<endl;
                                        for(i=0;i<3;i++)     {
                                         for(j=0;j<3;j++)     {
                                       cin>>t[i][j];
                                         }
                                        }
                                       cout<<"Translation matrix: "<<endl;
                                       for(i=0;i<3;i++)            {
                                        for(j=0;j<3;j++)            {
                                        cout<<t[i][j]<<"\t";
```

```
                                    }
                                    cout<<endl;     }

                                   for(i=0;i<3;i++) {
                                    for(j=0;j<3;j++)   {
                                     r1[i][j]=0;
                                     for(k=0;k<3;k++)           {
                                      r1[i][j]=r1[i][j]+a[i][k]*t[k][j];
                                     }
                                    }
                                   }
                                  cout<<"Resultant Translation matrix: "<<endl;
                                  for(i=0;i<3;i++)  {
                                   for(j=0;j<3;j++)  {
                                    cout<<r1[i][j]<<"\t";     }
                                     cout<<endl;      }
                                  x1=r1[0][0];
                                  y1=r1[0][1];
                                  x2=r1[1][0];
                                  y2=r1[1][1];
     line(x1,y1,x2,y2);
  }break;
  case 2:  {
   x1=a[0][0];
   y1=a[0][1];
   x2=a[1][0];
   y2=a[1][1];
     float ang,rad;
    cout<<"Enter the angle of rotation:"<<endl;
```

```
     cin>>ang;
    rad=ang*3.14/180;
    int x4,y4;
    x4=x2-(((x2-x1)*cos(rad))-((y2-y1)*sin(rad)));
    y4=y2-(((x2-x1)*sin(rad))+((y2-y1)*cos(rad)));
    line(x2,y2,x4,y4);
  }break;
 case 3:  {
    cout<<"Enter the value of scaling matrix"<<endl;
    for(i=0;i<3;i++)  {
      for(j=0;j<3;j++)  {
                                    cin>>s[i][j];
      }
    }
    cout<<"Scaling matrix: "<<endl;
     for(i=0;i<3;i++)    {
      for(j=0;j<3;j++)  {
                                    cout<<s[i][j]<<"\t";
      }
      cout<<endl;  }
   for(i=0;i<3;i++)  {
    for(j=0;j<3;j++) {
      r2[i][j]=0;
     for(k=0;k<3;k++)    {
      r2[i][j]=r2[i][j]+a[i][k]*s[k][j];
     }
    }
   }
```

```
    cout<<"Resultant Scaling matrix: "<<endl;
   for(i=0;i<3;i++)
   {
    for(j=0;j<3;j++)
    {
     cout<<r2[i][j]<<"\t";
    }
    cout<<endl;
   }
   x1=r2[0][0];
   y1=r2[0][1];
   x2=r2[1][0];
   y2=r2[1][1];
   line(x1,y1,x2,y2);
 }break;  }
 getch();
  closegraph();
  return 0;}
```

Output:



```
Enter the value of original matrix:
100 300 0
400 300 0
0   0   1
Original matrix:
100     300     0
400     300     0
0       0       1
*****MENU*****
Which type of transformation do you want to perform
1.Translation
2.Rotaion
3.Scaling
2
Enter the angle of rotation:
90
```

```
Enter the value of original matrix:
100 200 0
200 100 0
0   0   1
Original matrix:
100     200     0
200     100     0
0       0       1
*****MENU*****
Which type of transformation do you want to perform
1.Translation
2.Rotaion
3.Scaling
3
Enter the value of scaling matrix
2 0 0
0 2 0
0 0 1
Scaling matrix:
2       0       0
0       2       0
0       0       1
Resultant Scaling matrix:
200     400     0
400     200     0
0       0       1
```

Conclusion: We learnt about all types of 2D transformation i.e. translation, scaling and rotation and its matrices.

Questions :

1)What is translation tranformation?

2) What is shear transformation?

3)What is rotation transformation?

Experiment 9 : Program for polygon filling using boundary fill algorithm

Aim : Write a program for polygon filling using boundary fill algorithm

Theory :

Boundary Fill Algorithm starts at a pixel inside the polygon to be filled and paints the interior proceeding outwards towards the boundary. This algorithm works only if the color with which the region has to be filled and the color of the boundary of the region are different. If the boundary is of one single color, this approach proceeds outwards pixel by pixel until it hits the boundary of the region.
Boundary Fill Algorithm is recursive in nature. It takes an interior point(x, y), a fill color, and a boundary color as the input. The algorithm starts by checking the color of (x, y). If it's color is not equal to the fill color and the boundary color, then it is painted with the fill color and the function is called for all the neighbours of (x, y). If a point is found to be of fill color or of boundary color, the function does not call its neighbours and returns. This process continues until all points up to the boundary color for the region have been tested.
The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.
4-connected pixels : After painting a pixel, the function is called for four neighbouring points. These are the pixel positions that are right, left, above, and below the current pixel. Areas filled by this method are called 4-connected. Below given is the algorithm :

Algorithm :

```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
{
   if(getpixel(x, y) != boundary_color &&
     getpixel(x, y) != fill_color)
   {
      putpixel(x, y, fill_color);
      boundaryFill4(x + 1, y, fill_color, boundary_color);
      boundaryFill4(x, y + 1, fill_color, boundary_color);
      boundaryFill4(x - 1, y, fill_color, boundary_color);
      boundaryFill4(x, y - 1, fill_color, boundary_color);
   }
}
```

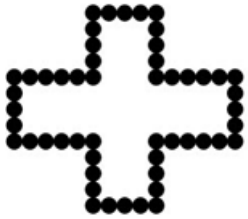


FIGURE BOUNDARY FILLED REGION

Code:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void flood(int,int,int,int);
void main()
{
   int gd=DETECT,gm;
   initgraph(&gd,&gm,"C:\\TURBOC3\\bgi");
   rectangle(50,50,100,100);
   flood(55,55,4,15);
   getch();
   closegraph();
}
void flood(int x,int y,int fillColor, int defaultColor)
{
   if((getpixel(x,y)!=defaultColor)&&(getpixel(x,y)!=fillColor))
   {
                                delay(1);
                                putpixel(x,y,fillColor);
                                flood(x+1,y,fillColor,defaultColor);
                                flood(x,y+1,fillColor,defaultColor);
                                flood(x-1,y,fillColor,defaultColor);
                                flood(x,y-1,fillColor,defaultColor);
   }
}
```

Output:

Conclusion: We studied the boundary fill algorithm using 4-connected approach and implemented it using cpp code and also learnt about 8-connected approach.

Questions :

1 Give types of algorithms for filling object with colors?

2. What is the difference between 4-connected approach and 8-connected approach?

3. What is the difference between boundary fill and flood fill approach?

Experiment 10 : Program for Line Clipping by Sutherland-Cohen

Aim : Write a Program for Line Clipping by Sutherland-Cohen

Theory:

Given a set of lines and a rectangular area of interest, the task is to remove lines that are outside the area of interest and clip the lines which are partially inside the area. Cohen-Sutherland algorithm divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are inside the given rectangular area.

ALGORITHM:

Step 1 : Assign a region code for two endpoints of given line.
Step 2 : If both endpoints have a region code 0000
then given line is completely inside.
Step 3 : Else, perform the logical AND operation for both region codes.
Step 3.1 : If the result is not 0000, then given line is completely outside.
Step 3.2 : Else line is partially inside.
Step 3.2.1 : Choose an endpoint of the line that is outside the given rectangle.
Step 3.2.2 : Find the intersection point of the rectangular boundary (based on region code).
Step 3.2.3 : Replace endpoint with the intersection point
and update the region code.
Step 3.2.4 : Repeat step 2 until we find a clipped line either
trivially accepted or trivially rejected.
Step 4 : Repeat step 1 for other lines

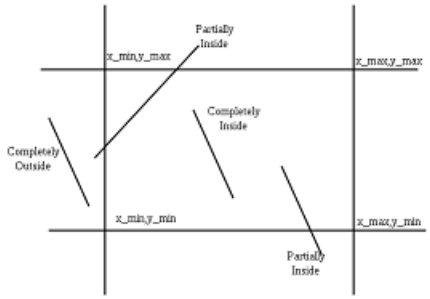


FIGURE THREE POSSIBLE CASES FOR ANY GIVEN LINE.

Code:

```
#include<stdio.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;
printf("\nEnter x1 and y1\n");
scanf("%d %d",&p1.x,&p1.y);
printf("\nEnter x2 and y2\n");
scanf("%d %d",&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);
```

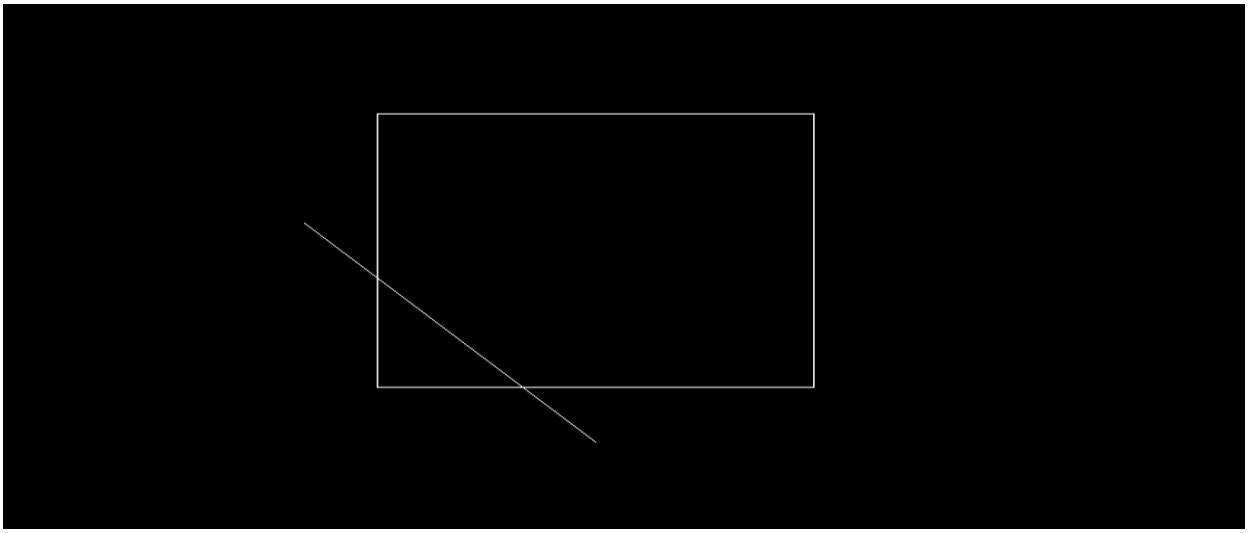
```
switch(v)
{
case 0: drawwindow();
delay(500);
drawline(p1,p2);
break;
case 1: drawwindow();
delay(500);
break;
case 2: p3=resetendpt(p1,p2);
p4=resetendpt(p2,p1);
drawwindow();
delay(500);
drawline(p3,p4);
break;
}
delay(5000);
closegraph();
}
 void drawwindow()
{
line(150,100,450,100);
line(450,100,450,350);
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);
```

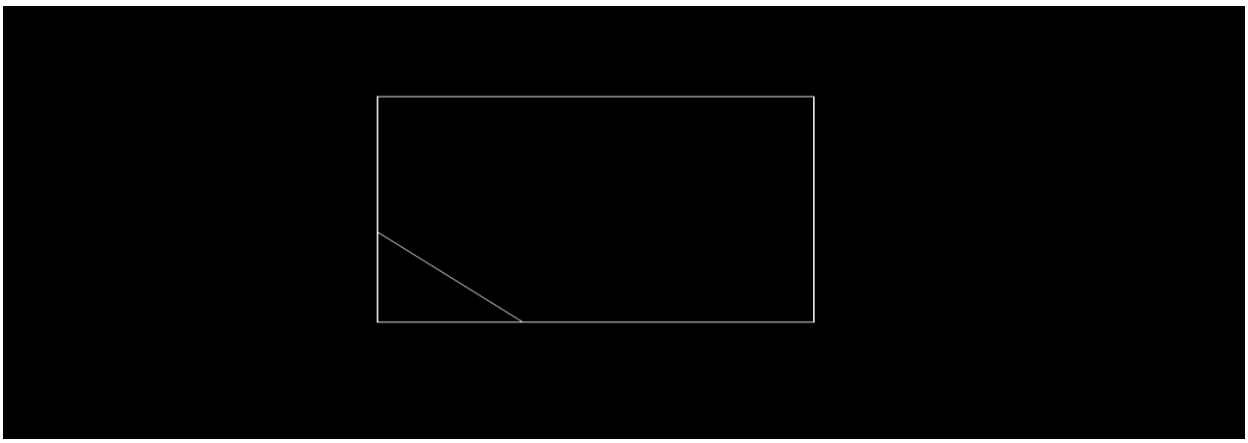
```
}
 PT setcode(PT p) //for setting the 4 bit code
{
PT ptemp;
if(p.y<100)
ptemp.code[0]='1'; //Top
else
ptemp.code[0]='0';
if(p.y>350)
ptemp.code[1]='1'; //Bottom
else
ptemp.code[1]='0';
if(p.x>450)
ptemp.code[2]='1'; //Right
else
ptemp.code[2]='0';
if(p.x<150)
ptemp.code[3]='1'; //Left
else
ptemp.code[3]='0';
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);
}
```

Output:





Conclusion: We learnt about the line clipping algorithm that helps us to understand how the invisible line are clipped.

Questions :

1. What are possible cases for a given line in Sutherland Cohen Algorithm?

2. How region codes are decided in Sutherland Cohen Algorithm?

Experiment 11: Program for Line Clipping by midpoint subdivision

Aim : Write a Program for Line Clipping by midpoint subdivision

Theory :

This algorithm is mainly used to compute visible areas of lines that are present in the view port are of the sector or the image. It follows the principle of the bisection method and works similarly to the Cyrus Beck algorithm by bisecting the line in to equal halves.
Like other algorithm, initially the line is tested for visibility. If line is completely visible it is drawn and if it is completely invisible it is rejected. If line is partially visible then it is subdivided in two equal parts. The visibility tests are then applied to each half. This subdivision process is repeated until we get completely visible and completely invisible line segments.

ALGORITHM:

Step1: Calculate the position of both endpoints of the line
Step2: Perform OR operation on both of these endpoints
Step3: If the OR operation gives 0000
then
Line is guaranteed to be visible
else
Perform AND operation on both endpoints.
If AND ≠ 0000
then the line is invisible
else
AND=6000
then the line is clipped case.
Step4: For the line to be clipped. Find midpoint
Xm=(x1+x2)/2
Ym=(y1+y2)/2
Xm is midpoint of X coordinate.
Ym is midpoint of Y coordinate.
Step5: Check each midpoint, whether it nearest to the boundary of a window or not.
Step6: If the line is totally visible or totally rejected not found then repeat step 1 to 5.
Step7: Stop algorithm.

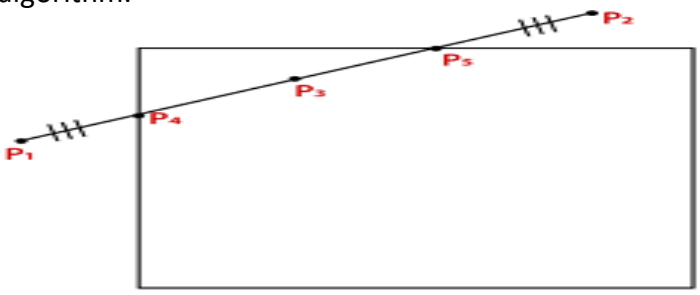


FIGURE BISECTION OF LINE IN MIDPOINT SUBDIVISION

Code:

```
#include <stdio.h>

#include <conio.h>
```

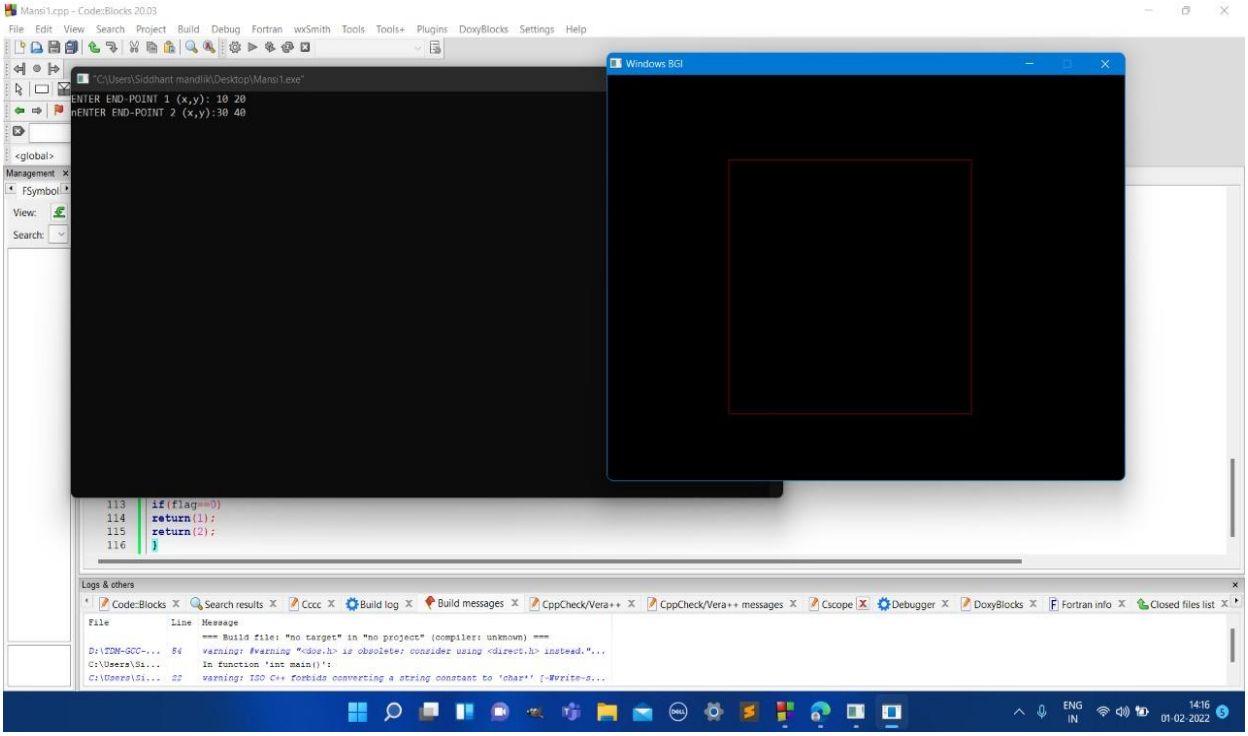
```
#include <stdlib.h>
#include <dos.h>
#include <math.h>
#include <graphics.h>
typedef struct coordinate
{
int x,y;
char code[4];
}PT;
void drawwindow();
void drawline (PT p1,PT p2);
PT setcode(PT p);
void midsub(PT p1,PT p2);
int visibility (PT p1,PT p2);
PT resetendpt (PT p1,PT p2);
main()
{
int gd=DETECT, gm,v;
PT p1,p2,ptemp;
initgraph(&gd,&gm,"c:\\tc\\bgi");
cleardevice();
printf("ENTER END-POINT 1 (x,y): ");
scanf("%d%d",&p1.x,&p1.y);
printf("nENTER END-POINT 2 (x,y):");
scanf("%d%d",&p2.x,&p2.y);
cleardevice();
drawwindow();
getch();
drawline(p1,p2);
```

```
getch();
cleardevice();
drawwindow();
midsub(p1,p2);
getch();
closegraph();
return(0);
}
void midsub(PT p1,PT p2)
{
PT mid;
int v;
p1=setcode(p1);
p2=setcode(p2);
v=visibility(p1,p2);
switch(v)
{
case 0:
drawline(p1,p2);
break;
case 1:break;
case 2:
mid.x = p1.x + (p2.x-p1.x)/2;
mid.y = p1.y + (p2.y-p1.y)/2;
midsub(p1,mid);
mid.x = mid.x+1;
mid.y = mid.y+1;
midsub(mid,p2);
break;}}
```

```
void drawwindow()
{
setcolor(RED);
line(150,100,450,100);
line(450,100,450,400);
line(450,400,150,400);
line(150,400,150,100);
}
void drawline (PT p1,PT p2)
{
setcolor(15);
line(p1.x,p1.y,p2.x,p2.y);
}
PT setcode(PT p)
{
PT ptemp;
if(p.y<=100) ptemp.code[0]='1'; else ptemp.code[0]='0'; if(p.y>=400)
ptemp.code[1]='1';
else
ptemp.code[1]='0';
if (p.x>=450)
ptemp.code[2]='1';
else
ptemp.code[2]='0';
if (p.x<=150)
ptemp.code[3]='1';
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);
}
```

Output:



Conclusion: We studied about the midpoint subdivision line clipping algorithm and implemented its code.

Questions :

1) What is viewport?
2) How does midpoint subdivision algorithm works?

Experiment 12 : Program for Text Generation

Aim : To Write a Program for Text Generation

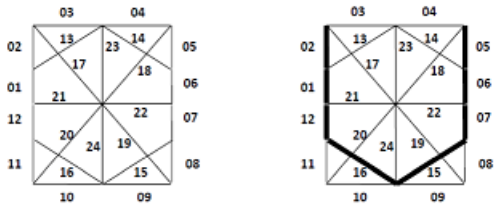
Theory

Letters, numbers, and other character are often displayed to label and annotate drawing and go give instructions and information to the user.Most of the times characters are built into the graphics display devices, usually as hardware but sometimes through software.

There are basic three methods for character generation:

Stroke method: This method uses small line segments to generate a character.
The small series of line segments are drawn like a strokes of a pen to form a character as shown in figure.

Starbust method: In this method a fix pattern of line segments are used to generate characters.As shown in figure, there are 24 line segments.Out of 24 line segments, segments required to display for particular character, are highlighted.This method is called starbust method because of its characteristic appearance.



Bitmap method: The third method for character generation.Also known as dot matrix because in this method characters are represented by an array of dots in the matrix form. It's a two dimensional array having columns and rows : 5 X 7 as shown in figure.7 X 9 and 9 X 13 arrays are also use
d.Higher resolution devices may use character array 100 X 100.

1	1	1	1	1	1	0
0	1	1	0	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	1	0
0	1	1	0	0	1	1
0	1	1	0	0	1	1
1	1	1	1	1	1	0

Code:

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
int main()
{
 int gdriver=DETECT,gmode;
 initgraph(&gdriver,&gmode,"C://TURBOC3//BGI");
 int a[7][9]={1,0,0,0,0,0,0,0,1,
    1,1,0,0,0,0,0,1,1,
    1,0,1,0,0,0,1,0,1,
    1,0,0,1,0,1,0,0,1,
    1,0,0,0,1,0,0,0,1,
    1,0,0,0,0,0,0,0,1,
    1,0,0,0,0,0,0,0,1
};
 int i,j,x,y;
 cout<<"The character is:"<<endl;
 for(i=0;i<7;i++)  {
  for(j=0;j<11;j++)   {
   if(a[i][j]==1)
   {
    putpixel(j+200,i+200,9);
   }
  } }
 getch();
 closegraph();
 return 0;
}
```

Output:

Conclusion: We studied bitmap text generation method that helps us to generate the text on the output screen.

Questions :

1) What is Stroke method?
2) What is Bit Map Method?
3) What is Star Bust Method?