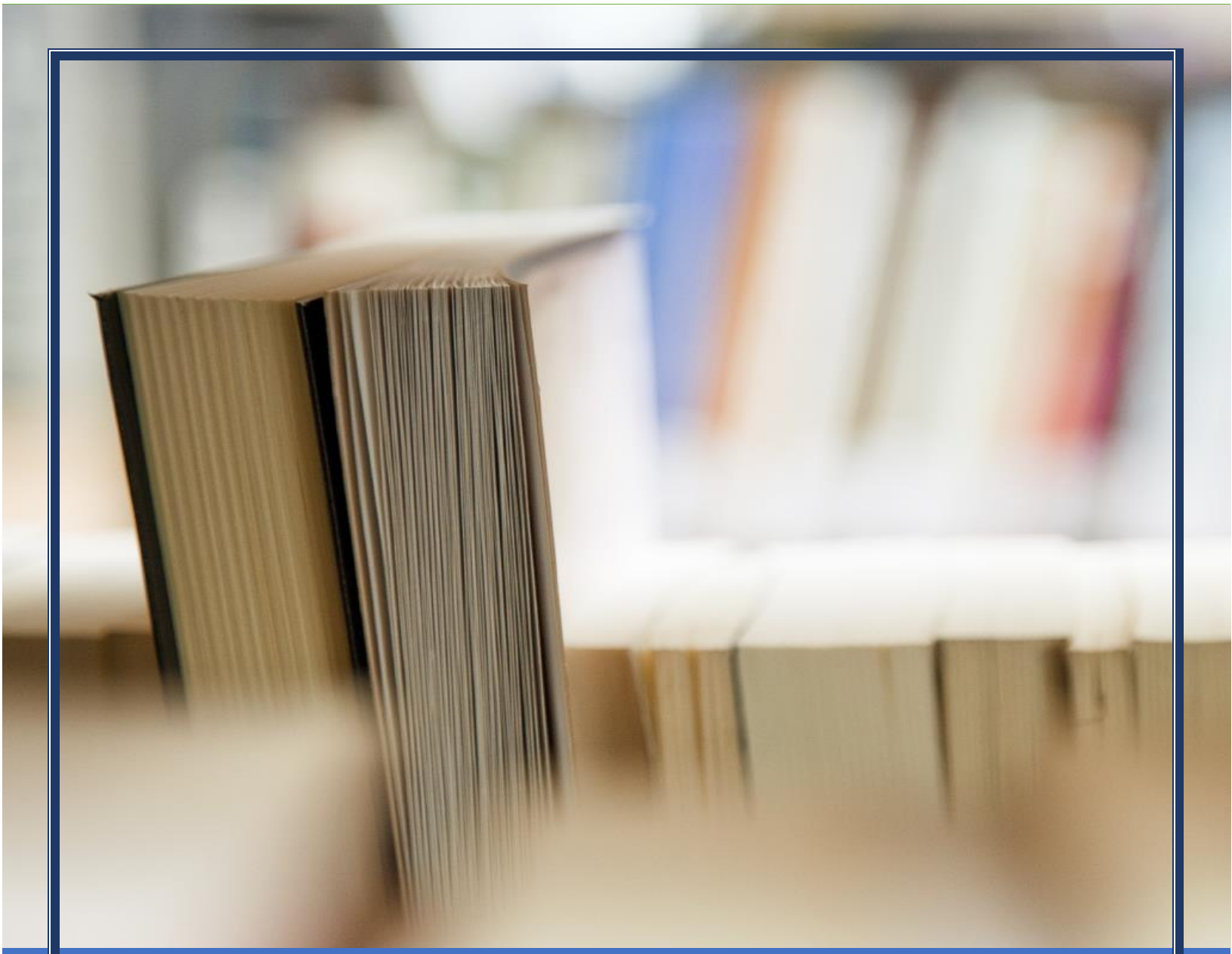


COMPUTER GRAPHICS &MULTIMEDIA LAB

NAME: ANAMIKA RAWAT

ENROLLMENT NO: 20220802720

BRANCH AND SECTION: CSE-C

SUBMITTED TO: Tina Dudeja Ma'am

INDEX

S.NO.	EXPERIMENT	REMARKS
1	Study and prepare list of graphic functions.	
2	To draw a Smiley using graphics header file in C/C++.	
3	Program for DDA Line Drawing Algorithm in C	
4	Write a C program to draw a line using Bresenham’s algorithm.	
5	Write a C program to draw a circle using mid-point algorithm.	
6	Implementation of Circle drawing algorithms using Bresenham's Algorithm.	
7	Write C Programs for the implementation of Bezier Curve	
8	Write C Programs for the implementation of Mid-point ellipse	

EXPERIMENT-1

Aim-: Study and prepare list of graphic functions.

Theory-: Different Types of Graphics Functions:

- **Line Function**

Line Function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line. The code given below draws a Line.

Syntax: line(int x1, int y1, int x2, int y2);

- **Rectangle Function**

Rectangle Function is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle. The code given below draws a Rectangle.

Syntax: rectangle(int left, int top, int right, int bottom);

- **Circle Function**

Circle Function is used to draw a circle with center (x,y) and third parameter specifies the radius of the circle. The code given below draws a Circle.

Syntax: circle(int x, int y, int radius);

- **Ellipse Function**

Ellipse Function is used to draw an ellipse (x,y) are coordinates of center of the ellipse, stangle is the starting angle, endangle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse. To draw a complete ellipse strangles and end angle should be 0 and 360 respectively. The code given below draws an Ellipse.

Syntax:

void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);

- **Initgraph Function**

Initgraph Function initializes the graphics system by loading a graphics driver from disk (or validating a registered driver), and putting the system into graphics mode.

Syntax: void initgraph(int *graphdriver, int *graphmode, char *pathtodriver);

- **Closegraph Function**

Closegraph Function closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called initgraph.

Syntax: void closegraph();

- **Getpixel Function**

Getpixel Function returns the color of the pixel present at the location (x,y).

Syntax: int getpixel(int x, int y);

- **Setcolor Function**

In Turbo Graphics each color is assigned a number. Total 16 colors are available. Strictly speaking number of available colors depends on current graphics mode and driver.

Syntax: void setcolor(int color);

- **Getcolor Function**

Getcolor Function returns the current drawing color.

Syntax: int getcolor();

- **Getbkcolor Function**

Getbkcolor Function returns the current background-color.

Syntax: int getbkcolor();_

EXPERIMENT-2

AIM: To draw a Smiley using graphics header file in C/C++.

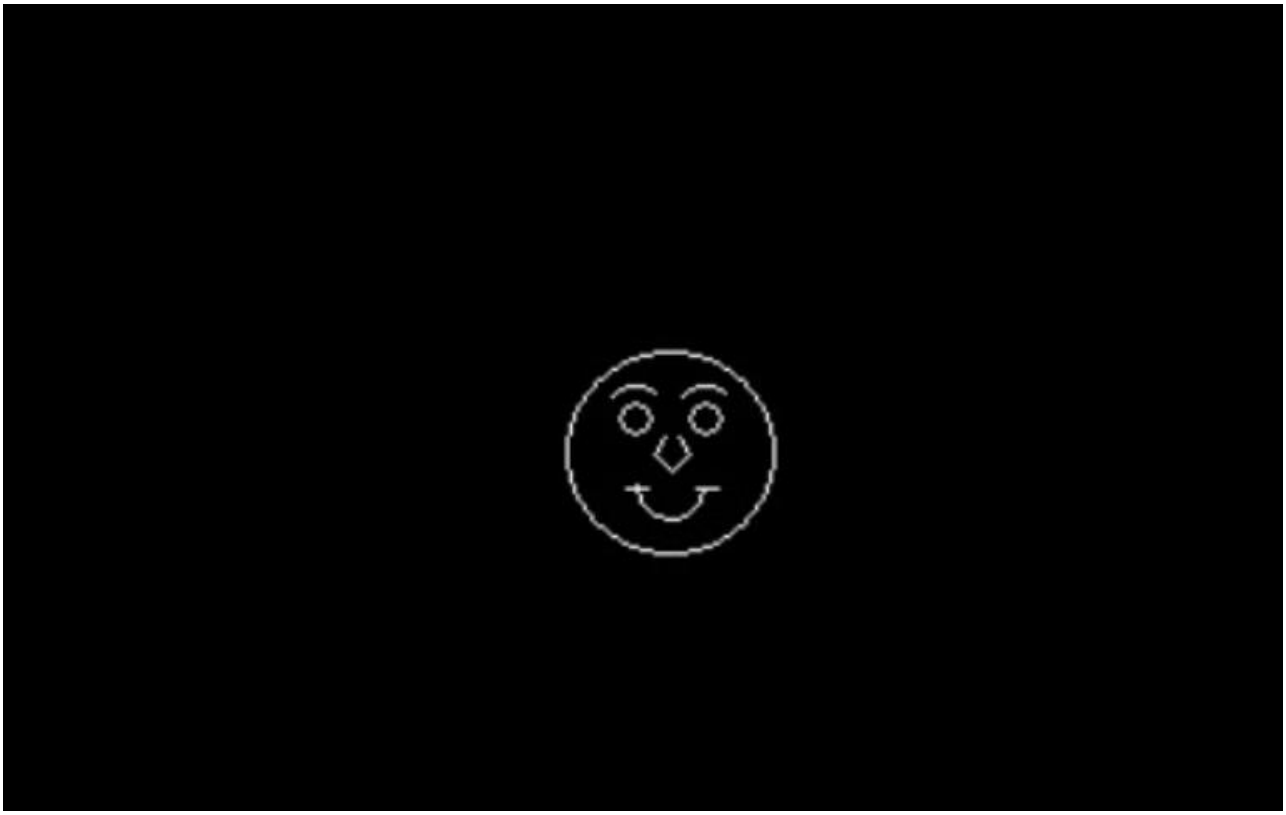
ALGORITHM AND CODE:

```
#include<graphics.h>
#include<stdio.h>
#include<conio.h>
int main()
{
  int gd = DETECT,gm;
  initgraph(&gd, &gm, "C:\\TC\\BGI");
//for head
  circle(200,200,30);
//for left eye
  circle(190,190,5);
  arc(190,190,50,130,10);
//for right eye
  circle(210,190,5);
  arc(210,190,50,130,10);
//for smiley lips
```

```
  arc(200,210,180,360,10);
  line(187,210,193,210);
  line(207,210,213,210);
//for nose
  line(198,195,195,200);
  line(202,195,205,200);
  line(195,200,200,205);
  line(205,200,200,205);
  #include<graphics.h>
#include<stdio.h>
#include<conio.h>
int main()
{
  int gd = DETECT,gm;
  initgraph(&gd, &gm, "C:\\TC\\BGI");
//for head
  circle(200,200,30);
//for left eye
  circle(190,190,5);
  arc(190,190,50,130,10);
```

```
//for right eye
  circle(210,190,5);
  arc(210,190,50,130,10);
//for smiley lips
  arc(200,210,180,360,10);
  line(187,210,193,210);
  line(207,210,213,210);
//for nose
  line(198,195,195,200);
  line(202,195,205,200);
  line(195,200,200,205);
  line(205,200,200,205);
  getch();
  closegraph();
}getch();
  closegraph();
}
```

OUTPUT:



EXPERIMENT-3

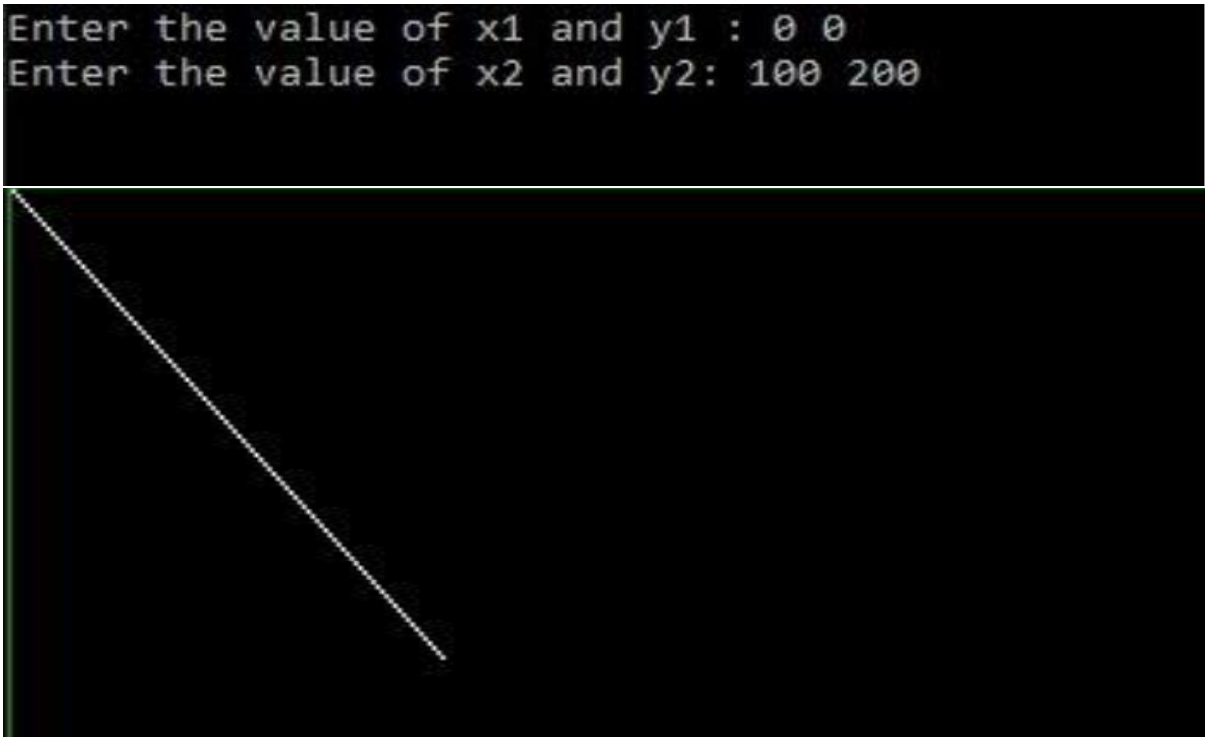
AIM: Program for DDA Line Drawing Algorithm in C

ALGORITHM:

```
#include<graphics.h>
#include<stdio.h>
#include<math.h>
int main()
{
int gd,gm,x,y,pixel,x1,x2,y1,y2,dx,dy;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"C://TurboC3//BGI");
printf("Enter the value of x1 : ");
scanf("%d",&x1);
printf("Enter the value of y1 : ");
scanf("%d",&y1);
printf("Enter the value of x2 : ");
scanf("%d",&x2);
printf("Enter the value of y2 : ");
scanf("%d",&y2);
dx=abs(x1-x2);
dy=abs(y1-y2);
if(dx>=dy)
pixel=dx;
else
pixel=dy;
dx=dx/pixel;
```

```
dy=dy/pixel;
x=x1;
y=y1;
while(pixel--)
{
putpixel(x,y,WHITE);
x=x+dx;
y=y+dy;
delay(100);
}
getch();
closegraph();
return 0;
}
```

OUTPUT:



EXPERIMENT-4

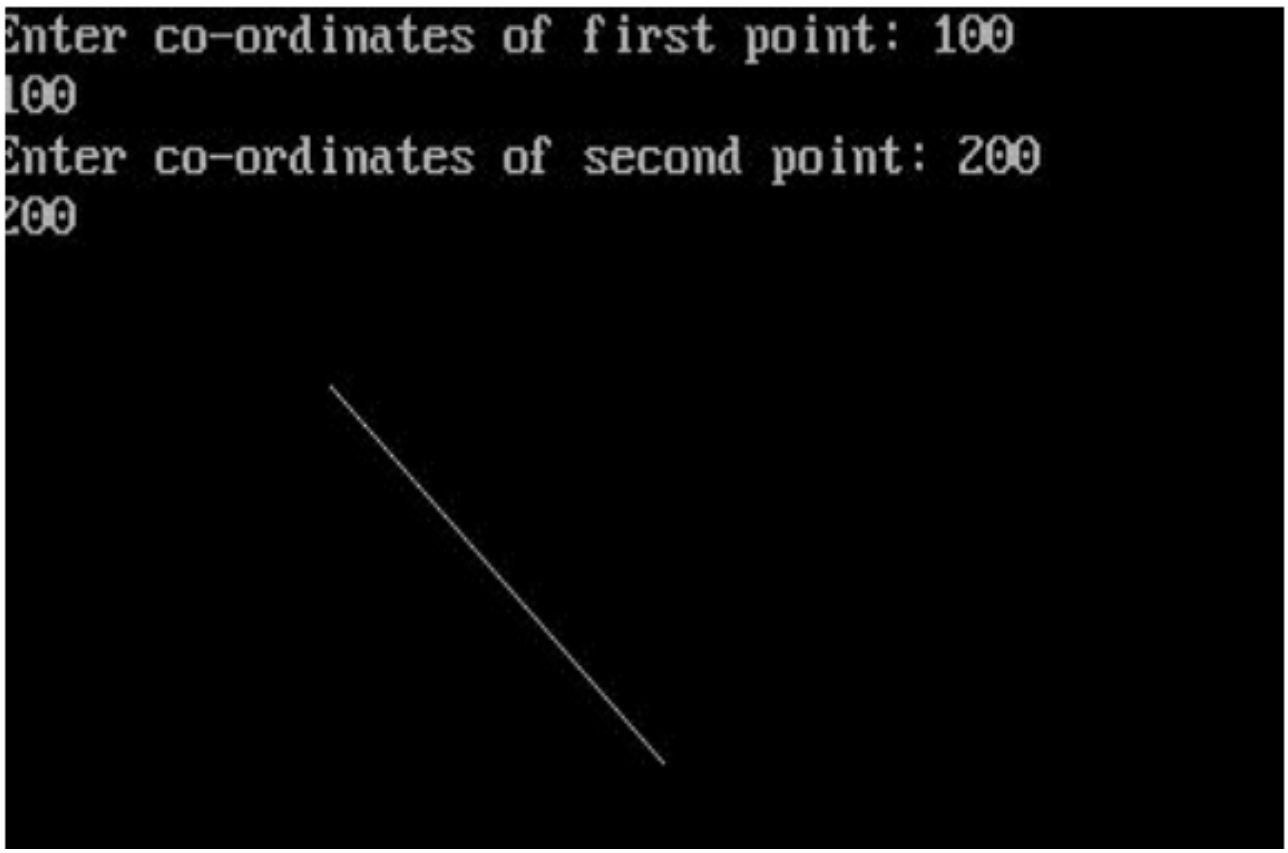
Aim-: Write a C program to draw a line using Bresenham's algorithm.

ALGORITHM AND CODE:

```
#include<stdio.h>
#include<graphics.h>
void drawline(int x0, int y0, int x1, int y1)
{
int dx, dy, p, x, y;
dx=x1-x0;
dy=y1-y0;
x=x0;
y=y0;
p=2*dy-dx;
while(x<x1)
{
if(p>=0)
{
putpixel(x,y,7);
y=y+1;
p=p+2*dy-2*dx;
}
else
{
putpixel(x,y,7);
p=p+2*dy;}
x=x+1;
```

```
}
}
int main()
{
int gdriver=DETECT, gmode, error, x0, y0, x1, y1;
initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
printf("Enter co-ordinates of first point: ");
scanf("%d%d", &x0, &y0);
printf("Enter co-ordinates of second point: ");
scanf("%d%d", &x1, &y1);
drawline(x0, y0, x1, y1);
return 0;
}
```

OUTPUT :



EXPERIMENT-5

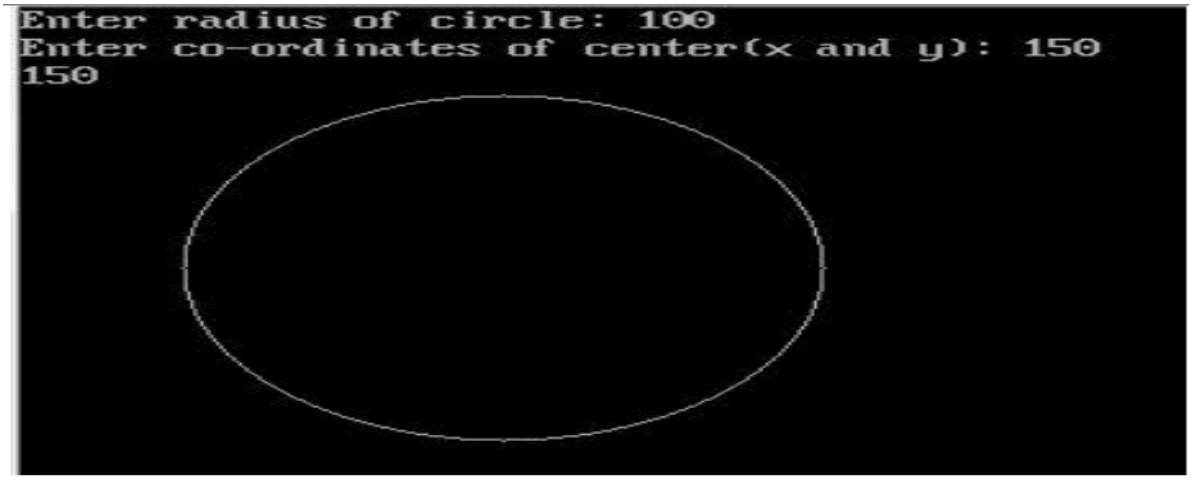
Aim-: Write a C program to draw a circle using mid-point algorithm.

ALGORITHM AND CODE:

```
#include<stdio.h>
#include<graphics.h>
void midPointCircleDraw(int x_centre, int y_centre, int r)
{
int x = r, y = 0;
printf("(%d, %d) ", x + x_centre, y + y_centre);
if (r > 0)
{
printf("(%d, %d) ", x + x_centre, -y + y_centre);
printf("(%d, %d) ", y + x_centre, x + y_centre);
printf("(%d, %d)\n", -y + x_centre, x + y_centre);
}
int P = 1 - r;
while (x > y)
{
y++;
if (P <= 0)
P = P + 2*y + 1;
else
{
x--;
P = P + 2*y - 2*x + 1;
}
```

```
if (x < y)
break;
printf("(%d, %d) ", x + x_centre, y + y_centre);
printf("(%d, %d) ", -x + x_centre, y + y_centre);
printf("(%d, %d) ", x + x_centre, -y + y_centre);
printf("(%d, %d)\n", -x + x_centre, -y + y_centre);
if (x != y)
{
printf("(%d, %d) ", y + x_centre, x + y_centre);
printf("(%d, %d) ", -y + x_centre, x + y_centre);
printf("(%d, %d) ", y + x_centre, -x + y_centre);
printf("(%d, %d)\n", -y + x_centre, -x + y_centre);
}
}
}
int main()
{
midPointCircleDraw(0, 0, 3);
return 0;
}
```

OUTPUT :



EXPERIMENT-6

Aim-: Implementation of Circle drawing algorithms using Bresenham's Algorithm.

ALGORITHM AND CODE:

```
#include <stdio.h>
#include <dos.h>
#include <graphics.h>
void drawCircle(int xc, int yc, int x, int y)
{
putpixel(xc+x, yc+y, RED);
putpixel(xc-x, yc+y, RED);
putpixel(xc+x, yc-y, RED);
putpixel(xc-x, yc-y, RED);
putpixel(xc+y, yc+x, RED);
putpixel(xc-y, yc+x, RED);
putpixel(xc+y, yc-x, RED);
putpixel(xc-y, yc-x, RED);
}
void circleBres(int xc, int yc, int r)
{
int x = 0, y = r;
int d = 3 - 2 * r;
drawCircle(xc, yc, x, y);
while (y >= x)
{
x++;
if (d > 0)
```

```
{
y--;
d = d + 4 * (x - y) + 10;
}
else
d = d + 4 * x + 6;
drawCircle(xc, yc, x, y);
delay(50);
}
}
int main()
{
int xc = 50, yc = 50, r2 = 30;
int gd = DETECT, gm;
initgraph(&gd, &gm, "");
circleBres(xc, yc, r);
return 0;
}
```

OUTPUT :

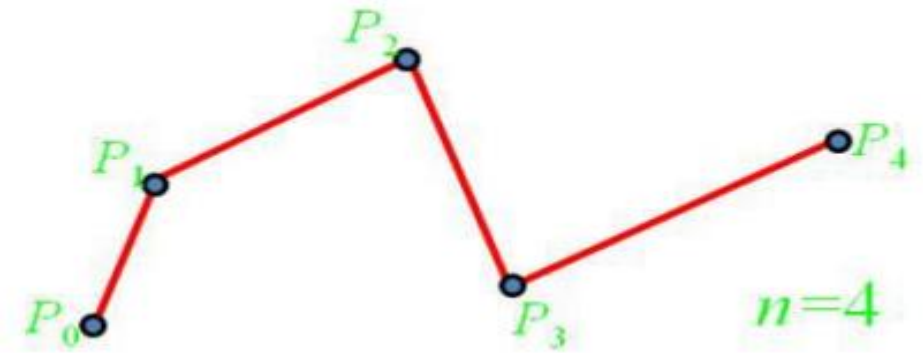
EXPERIMENT-7

Aim-: -: Write C Programs for the implementation of Bezier Curve.

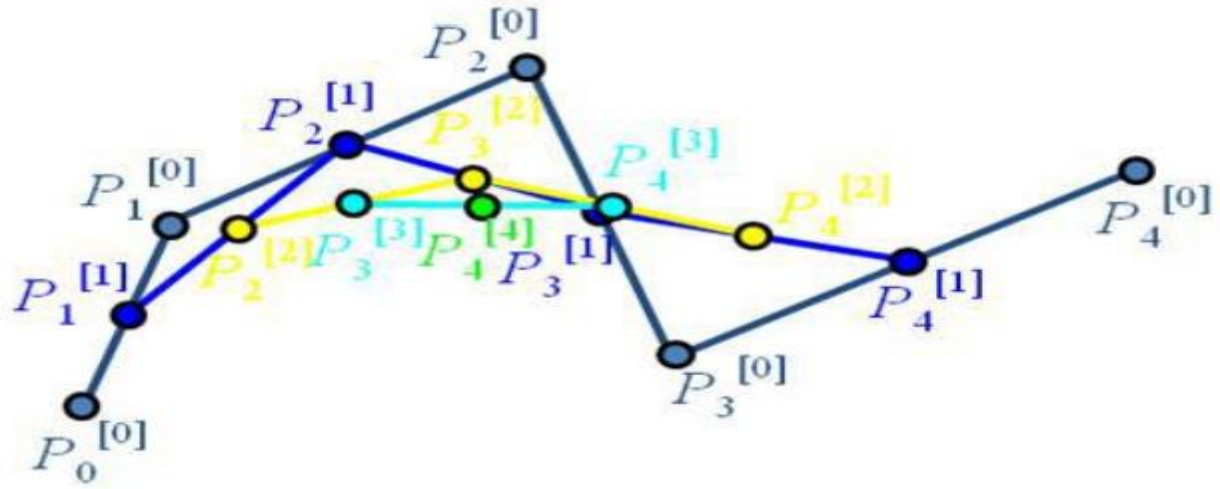
ALGORITHM AND CODE:

Constructive Bezier Curve Algorithm

Consider the n+1 points P0,…,Pn and connect the points into a polyline we will denote hereafter as the control polygon.



Given points Pi, i = 0,...,n, our goal is to determine a curve g (t), for all values tÎ [0,1]. The idea is demonstrated below:



Basic Algorithm

The objective here is to find points in the middle of two nearby points and iterate this until we have no more iterations. The new values of points will give us the curve. The famous Bezier equation is the exact formulation of this idea. Here is the algorithm:

Step 1: Select a value t Î [0,1]. This value remains constant for the rest of the steps.

Step 2: Set Pi[0] (t) = Pi, for i = 0,...,n.

Step 3: For j= 0,...,n, set for i = j,...,n.

Step 4: g (t) = Pn[n] (t)

Special & General Cases

Now, I will give formulas for common, special cases that can be helpful in certain applications. The code of the article does not demonstrate any of them, but it uses the generalized formula. So, let me start with the generalized formula:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} \mathbf{P}_i (1-t)^{n-i} t^i = \mathbf{P}_0 (1-t)^n + \binom{n}{1} \mathbf{P}_1 (1-t)^{n-1} t + \cdots + \mathbf{P}_n t^n \ , \ t \in [0,1].$$

For the sake of simplicity and convention used in this article and code, it is better to represent this formula as:

$$\gamma(t) = \sum_{i=0}^{n} P_i \frac{n!}{i!(n-i)!} (1-t)^{n-i} t^i \ ,$$

What this equation tells us is nothing but the formulation of the above algorithm (the mid-point iterations). It is very important in the sense that a whole algorithm could be summarized into a formula and a straightforward implementation would yield

correct results. Here, n denotes the number of points and P denotes the points themselves. The factorial coefficients of the points are simply called the Bernstein basis functions, because of the name of the founder.

Here are the special cases:

Linear Bezier:

$$\mathbf{B}(t) = \mathbf{P}_0 + (\mathbf{P}_1 - \mathbf{P}_0)t = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , t \in [0,1]$$

Quadratic Bezier:

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2 \ , t \in [0,1].$$

Cubic Bezier:

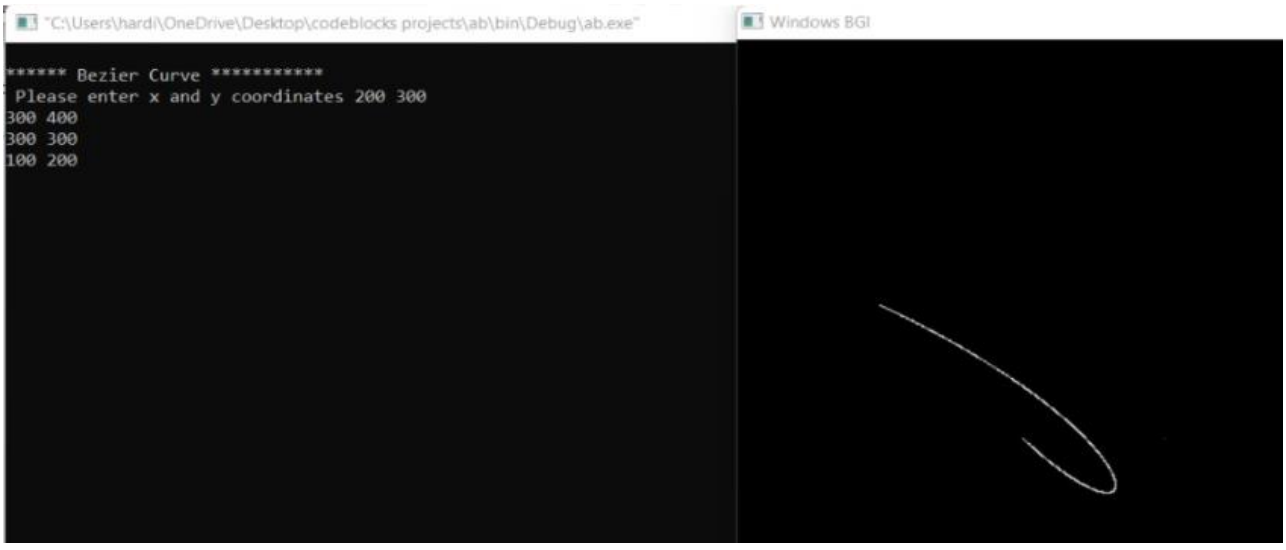
$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3 \ , t \in [0,1].$$

CODE:

```
#include<bits/stdc++.h>
#include<graphics.h>
#include<math.h>
#include<conio.h>
#include<stdio.h>
using namespace std;
int main()
{
int x[4],y[4],i;
double put_x,put_y,t;
int gr=DETECT,gm;
```

```
initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");
printf("\n*** Bezier Curve ****");
printf("\n Please enter x and y coordinates ");
for(i=0;i<4;i++)
{
scanf("%d %d",&x[i],&y[i]);
putpixel(x[i],y[i],3);
}
for(t=0.0;t<=1.0;t=t+0.001)
{
put_x = pow(1-t,3)x[0] + 3*t*pow(1-t,2)*x[1] + 3*t*t(1-t)*x[2] +
pow(t,3)*x[3];
put_y = pow(1-t,3)y[0] + 3*t*pow(1-t,2)*y[1] + 3*t*t(1-t)*y[2] +
pow(t,3)*y[3];
putpixel(put_x,put_y, WHITE);
}
getch();
closegraph();
}
```

OUTPUT



EXPERIMENT-8

Aim-: -: Write C Programs for the implementation of Mid-point ellipse.

CODE:

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

 void ellipse(int xc,int yc,int rx,int ry)
 {
   int gm=DETECT,gd;
   int x, y, p;
   clrscr();
 initgraph(&gm,&gd,"C:\\TC\\BGI");
   x=0;
   y=ry;
   p=(ry*ry)-(rx*rx*ry)+((rx*rx)/4);
   while((2*x*ry*ry)<(2*y*rx*rx))
   {
       putpixel(xc+x,yc-y,WHITE);
       putpixel(xc-x,yc+y,WHITE);
       putpixel(xc+x,yc+y,WHITE);
       putpixel(xc-x,yc-y,WHITE);

       if(p<0)
       {
     x=x+1;
```

```
    p=p+(2*ry*ry*x)+(ry*ry);
      }
      else
      {
    x=x+1;
    y=y-1;
    p=p+(2*ry*ry*x+ry*ry)-(2*rx*rx*y);
      }
  }
  p=((float)x+0.5)*((float)x+0.5)*ry*ry+(y-1)*(y-1)*rx*rx-
rx*rx*ry*ry;

       while(y>=0)
  {
      putpixel(xc+x,yc-y,WHITE);
      putpixel(xc-x,yc+y,WHITE);
      putpixel(xc+x,yc+y,WHITE);
      putpixel(xc-x,yc-y,WHITE);

      if(p>0)
      {
    y=y-1;
    p=p-(2*rx*rx*y)+(rx*rx);

      }
      else
      {
    y=y-1;
    x=x+1;
```

```
    p=p+(2*ry*ry*x)-(2*rx*rx*y)-(rx*rx);
      }
  }
  getch();
  closegraph();
}

void main()
{
  int xc,yc,rx,ry;
  clrscr();
  printf("Enter Xc=");
  scanf("%d",&xc);
  printf("Enter Yc=");
  scanf("%d",&yc);
  printf("Enter Rx=");
  scanf("%d",&rx);
  printf("Enter Ry=");
  scanf("%d",&ry);
  ellipse(xc,yc,rx,ry);
  getch();
}
```

OUTPUT