# **Lab 1: Introduction to graphics primitive and graphics drivers**

1. Software requirement: Turbo C / C++

**BASIC GRAPHICS FUNCTION**

1. **INITGRAPH**

• Initializes the graphics system.

**Declaration**

• Void far initgraph(int far \*graphdriver)

**Remarks**

• To start the graphic system, you must first call initgraph.

• Initgraph initializes the graphic system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode.

• Initgraph also resets all graphics settings (color, palette, current position, viewport, etc) to their defaults then resets graph.

1. **GETPIXEL, PUTPIXEL**

• Getpixel gets the color of a specified pixel.

• Putpixel places a pixel at a specified point.

**Decleration**

• Unsigned far getpixel(int x, int y)

• Void far putpixel(int x, int y, int color)

**Remarks**

• Getpixel gets the color of the pixel located at (x,y);

• Putpixel plots a point in the color defined at (x, y)

**Return value**

• Getpixel returns the color of the given pixel.

• Putpixel does not return

1. **CLOSE GRAPH**

• Shuts down the graphic system.

**Declaration**

• Void far closegraph(void);

**Remarks**

• Close graph deallocates all memory allocated by the graphic system.

• It then restores the screen to the mode it was in before you called initgraph.

**Return value**

• None.

1. **ARC, CIRCLE, PIESLICE**

• arc draws a circular arc.

• Circle draws a circle

• Pieslice draws and fills a circular pieslice

**Declaration**

• Void far arc(int x, int y, int stangle, int endangle, int radius);

• Void far circle(int x, int y, int radius);

• Void far pieslice(int x, int y, int stangle, int endangle, int radius);

**Remarks**

• Arc draws a circular arc in the current drawing color

• Circle draws a circle in the current drawing color

• Pieslice draws a pieslice in the current drawing color, then fills it using the current fill pattern and fill color.

1. **ELLIPSE, FILL ELIPSE, SECTOR**

• Ellipse draws an elliptical arc.

• Fill ellipse draws and fills ellipse.

• Sector draws and fills an elliptical pie slice.

**Declaration**

• Void far ellipse (int x, int y, int stangle, int endangle, int xradius, int yradius)

• Void far fill ellipse (int x, int y, int xradius, int yradius)

• Void farsectoe(int x, int y, int stangle, int endangle, int xradius, int yradius)

**Remarks**

• Ellipse draws an elliptical arc in the current drawing color.

• Fill ellipse draws an elliptical arc in the current drawing color and then fills it with fill color and fill pattern.

• Sector draws an elliptical pie slice in the current drawing color and then fills it using the pattern and color defined by setfill style or setfill pattern.

1. **FLOODFILL**

• Flood-fills a bounded region.

**Declaration**

• Void far floodfill(int x, int y, int border)

**Remarks**

• Floodfills an enclosed area on bitmap device.

• The area bounded by the color border is flooded with the current fill pattern and fill color.

• (x,y) is a “seed point”

¬ If the seed is within an enclosed area, the inside will be filled.

¬ If the seed is outside the enclosed area, the exterior will be filled.

• Use fillpoly instead of floodfill wherever possible so you can maintain code compatibility with future versions.

• Floodfill doesnot work with the IBM-8514 driver.

**Return value**

• If an error occurs while flooding a region, graph result returns „1‟.

**7) GETCOLOR, SETCOLOR**

• Getcolor returns the current drawing color.

• Setcolor returns the current drawing color.

**Declaration**

• Int far getcolor(void);

• Void far setcolor(int color)

**Remarks**

• Getcolor returns the current drawing color.

• Setcolor sets the current drawing color to color, which can range from 0 to getmaxcolor.

• To set a drawing color with set color, you can pass either the color number or the equivalent color name.

1. **LINE, LINEREL, LINETO**

• Line draws a line between two specified pints.

• Onerel draws a line relative distance from current position (CP).

• Linrto draws a line from the current position (CP) to(x,y).

• Void far lineto(int x, int y)

**Remarks**

• Line draws a line from (x1, y1) to (x2, y2) using the current color, line style and thickness. It does not update the current position (CP).

• Linerel draws a line from the CP to a point that is relative distance (dx, dy) from the CP, then advances the CP by (dx, dy).

• Lineto draws a line from the CP to (x, y), then moves the CP to (x,y).

**Return value**

• None

1. **RECTANGLE**

• Draws a rectangle in graphics mode.

**Decleration**

• Void far rectangle (int left, int top, int right, int bottom)

**Remarks**

• It draws a rectangle in the current line style, thickness and drawing color.

• (left, top) is the upper left corner of the rectangle, and (right, bottom) is its lower right corner.

**Return value**

# **LAB 2: Implementation of line drawing algorithms – DDA(Digital Differential Algorithm)**

**Algorithm:**

Step 1. Declare the variables, x1,y1 and x2 , y2 dx, dy ,del x, del y as real and k as integer.

Step 2. Perform

dx = x2-x1

dy = y2 – y1

Step 3. Test if |dy|<|dx| then

Steps = |dx|

Else steps = |dy|

Step 4. set del x = dx/steps

del y = dy/steps

x= x1

y = y1

Step 5. Plot (x, y)

Step 6. Do for k = 1 to steps

x = x+ delx

y = y +del y

Plot (x,y)

# **Program using C language:**

#include<graphics.h>

#include<conio.h>

#include<stdio.h>

void main()

{

int gd = DETECT ,gm, i;

float x, y,dx,dy,steps;

int x0, x1, y0, y1;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

setbkcolor(WHITE);

x0 = 100 , y0 = 200, x1 = 500, y1 = 300;

dx = (float)(x1 - x0);

dy = (float)(y1 - y0);

if(dx>=dy)

{

steps = dx;

}

else

{

steps = dy;

}

dx = dx/steps;

dy = dy/steps;

x = x0;

y = y0;

i = 1;

while(i<= steps)

{

putpixel(x, y, RED);

x += dx;

y += dy;

i=i+1;

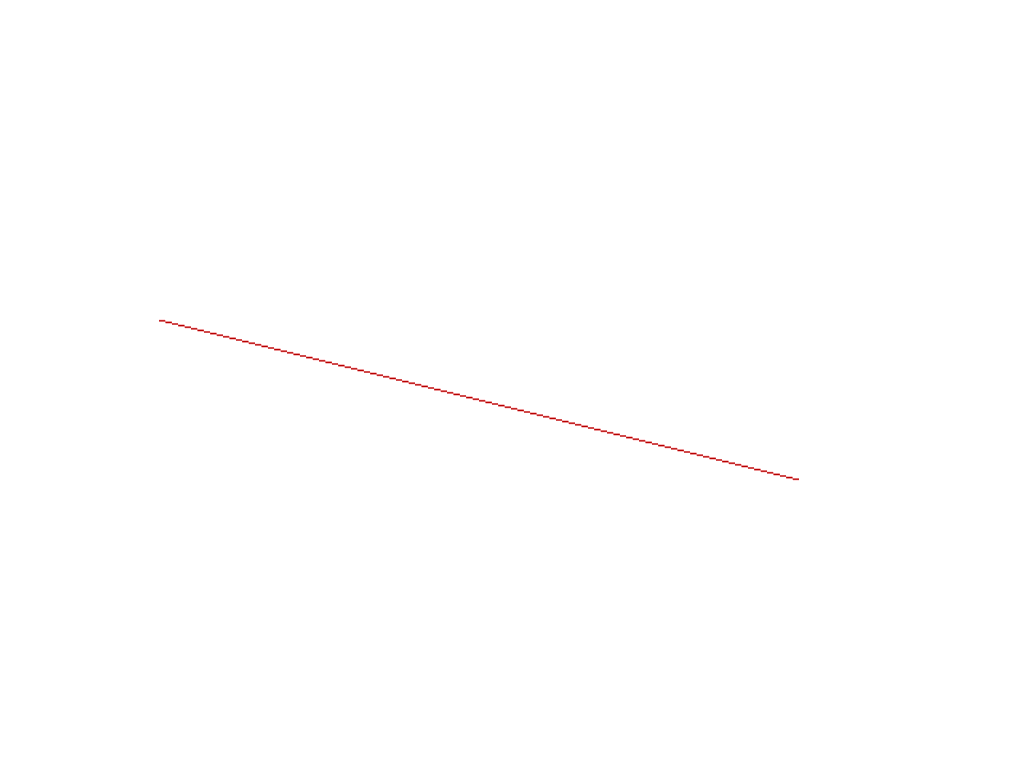
}

getch();

closegraph();

}

**Output:**



## **LAB 3: Implementation of line drawing algorithms – Bresenham's Line Algorithm**

**Algorithm:**

**Step1:** Start Algorithm

**Step2:** Declare variable x1, x2, y1, y2, d, i1, i2, dx, dy

**Step3:** Enter value of x1, y1, x2, y2  
                Where x1, y1 are coordinates of starting point  
                And x2, y2 are coordinates of Ending point

**Step4:** Calculate dx = x2-x1  
                Calculate dy = y2-y1  
                Calculate i1=2 \* dy  
                Calculate i2=2 \* (dy - dx)  
                Calculate d=i1 - dx

**Step5:** Consider (x, y) as starting point and xend as maximum possible value of x.  
                If dx < 0  
                        Then x = x2  
                        y = y2  
                        xend = x1  
                If dx > 0  
                    Then x = x1  
                y = y1  
                        xend = x2

**Step6:** Generate point at (x, y) coordinates.

**Step7:** Check if whole line is generated.  
                If x > = xend  
                Stop.

**Step8:** Calculate co-ordinates of the next pixel  
                If d < 0  
                    Then d = d + i1  
                If d ≥ 0  
          Then d = d + i2  
                Increment y = y + 1

**Step9:** Increment x = x + 1

**Step10:** Draw a point of latest (x, y) coordinates

**Step11:** Go to step 7

**Step12:** End

# **Program using C language:**

#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);

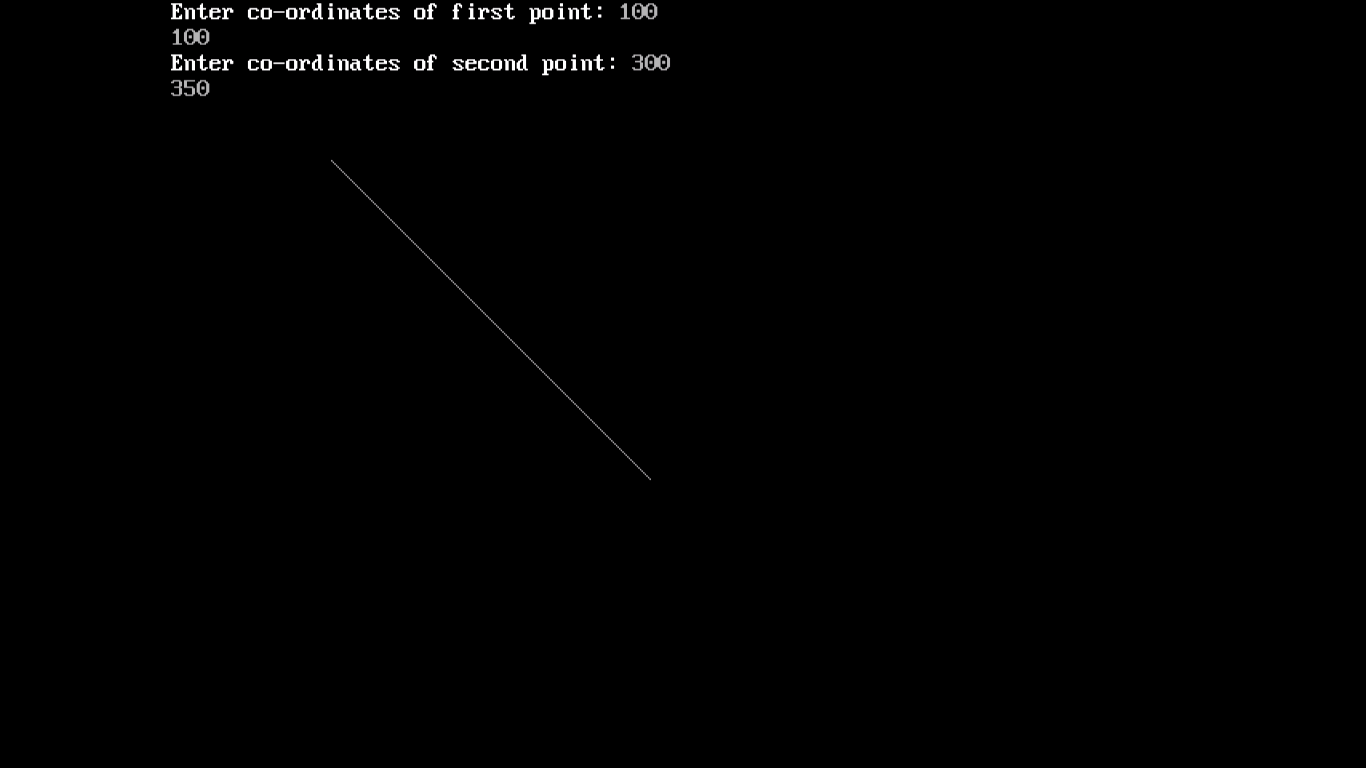
printf("Enter co-ordinates of second point: ");

scanf("%d%d", &x1, &y1);

drawline(x0, y0, x1, y1);

return 0;

}

**Output:**