SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

PRACTICAL-1

AIM: To study the various graphics commands in C language.

Description:

In the C programming language, graphics commands are used to create graphical applications and manipulate graphical elements on the screen. Graphics programming in C often involves using a graphics library or API (Application Programming Interface) that provides functions for drawing shapes, lines, text, and other graphical elements.

One popular graphics library for C is the "graphics.h" library, which is part of the Borland Graphics Interface (BGI). This library provides a set of functions for basic graphics operations and is commonly used for simple graphics programming in C.

Here are some common graphics commands used in the "**graphics.h**" library:

initgraph(): Initializes the graphics system and creates a graphics window.
closegraph(): Closes the graphics system and releases resources.
line(): Draws a line between two points.
circle(): Draws a circle with a specified center and radius.
rectangle(): Draws a rectangle with specified coordinates for the top-left and bottom-right corners.
bar(): Draws a filled rectangle.
outtextxy(): Displays text at a specified location on the screen.
setcolor(): Sets the current drawing color.
getch(): Waits for a keypress.

Post Practical Questions:

1. **Command Understanding:**
   Select any three graphics commands used in your program and explain their significance in computer graphics. Discuss how they contribute to creating a graphical interface and mention any specific parameters they require.

Ans: 1) Draw line 2) Draw Rectangle 3) Draw text 4) The draw line command is used to create straight line between two points on the screen It's Fundamental command in computer graphics because lines are the building blocks of many shapes and forms By specifying the coordinates of the start and end points of the line along parameter such as colour and thickness drawline allows for creation of various shapes

2. **Application Scenarios:**
Describe a real-world scenario where understanding and using graphics commands would be crucial. Explain how specific graphics commands you've studied in this practical could be applied in that scenario.

Ans: One real world scenario where understanding using graphics commands is critical is in the field of computer aided design In CAD software engineer and designs create, detailed digital models of product buildings or mechanical components before they are manufactured or constructed in the physical world,

3. **Alternative Approaches:**
Investigate alternative methods or graphics libraries that provide similar functionalities to the commands you used. Compare and contrast these alternatives with the graphics commands in terms of ease of use, performance, and flexibility.

Ans: 1) OPen GL 2) WPBORTGL : is a Java Scpcit API. 1). is a course platform API. for repoding 2D and 3D vector graphics It provide wide range of funation for reporaiding, poimitive such as paints, line and palygons as well as. more advums GL is widely used in various applicatin oranging from video games to specific visulization.

Conclusion:

In this practicle we know the various command s'on C denguage.

Marks out of 10	9
Signature with Date of Completion	

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

PRACTICAL - 2

AIM: To Develop the DDA Line drawing algorithm using C language

Description:

The DDA (Digital Differential Analyzer) algorithm is a simple and efficient method for drawing a straight line between two given points on a raster display device. It uses the concept of incremental changes along the x and y axes to determine the pixels that the line passes through. Here's a detailed explanation of the DDA Line drawing algorithm:

Algorithm:

Input:

- (x1, y1): Coordinates of the starting point.
- (x2, y2): Coordinates of the ending point.

Calculate Differences:

- Calculate the differences between the x and y coordinates: dx = x2 - x1 and dy = y2 - y1.

Determine the Number of Steps:

- Find the number of steps needed to reach from (x1, y1) to (x2, y2) by taking the maximum absolute difference between dx and dy. Let steps be the maximum of |dx| and |dy|.

Calculate Increment Values:

- Calculate the increments along the x and y axes. Let xIncrement = dx / steps and yIncrement = dy / steps.

Draw the Line:

- Use a loop that iterates steps times to draw the line. In each iteration, update the current coordinates (x, y) and plot the pixel at (round(x), round(y)).

Here, round(x) and round(y) represent rounding to the nearest integer.
The loop essentially increments the x and y values in small steps, ensuring that the line drawn covers the entire distance from (x1, y1) to (x2, y2).

Coding:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
void main()
{
    int i, gd, gm;
    float x, y, x1, y1, x2, y2, x_inca,
    y_inca, length, dx, dy, m;
```

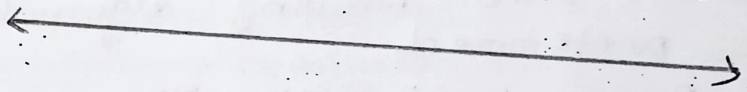
```
printf("Enter the value of x1=");
scanf
      ("%f",&x1);
printf("Enter the value: x2 = ");
scanf("%f",&x2);
printf("Enter the value: y1 = ");
scanf("%f",&y1);
printf("Enter the value y2 = ");
scanf("%f",&y2);
detectgraph(&gd,&gm);
initgraph(&gd,&gm," ");
initgraph(&gd,&gm," ");
setcolor(7);
dx = abs(x2-x1)
dy = abs(y2-y1)
m = dy/dx;
printf("slope = %f", m
if(dx==dy){
    length=dx;
}else{
    length = dy.
}
x_incr = abs(x2-x1)/length;
y_incr = abs(y2-y1)/length;
x = x1;
y = y1;
putpixel( x, y, y2)
i = 1
while(i<=length)
```

```
    x = x + x_incr;
    y = y + y_incr;
    putpixel(x, y, 12);
    printf("\n%f %f", x, y);
    i = i + 1;
  delay(50);
}
getch();
closegraph();
```

OUTPUT

Enter first coordinates : 100 150

Enter second coordinates : 125 175



Post Practical Questions:

1. **Algorithm Understanding:**

   Provide a step-by-step explanation of the DDA Line drawing algorithm implemented in your program. Discuss its advantages and limitations compared to other line drawing algorithms.

Ans: input : The algorithm takes two endpoints of the line segment (x1, y1) and (x2, y2) as input.

calculate the difference

$\Delta x = x2 - x1$ $\Delta x\_step = \Delta x / steps$

$\Delta y = y2 - y1$ $\Delta y\_step = \Delta y / steps$

2. **Precision and Efficiency:**

   DDA can suffer from precision issues. Discuss strategies to enhance precision and efficiency in the DDA Line drawing algorithm. Consider the impact of rounding errors and suggest possible improvements.

Ans: Calculate differences $\Delta x = x2 - x1$ $\Delta y = y2 - y1$.

Determine steps :- $N = \max(|\Delta x1|, |\Delta y1|)$

Calculate increment values. $\Delta x\_inc = \Delta x / N$

End After N steps line is drawn from (x1, y1) to (x2, y2)

Advantage : 1) Simplicity 2) efficiency 3) Straightforward

Limitation : 1) Precious 2) inefficiency in memory usage

3. **Practical Applications:**
Identify and discuss real-world applications where the DDA Line drawing algorithm would be useful. Explain why this algorithm is suitable for these applications and if there are scenarios where other line drawing algorithms might be preferred.

Ans: 1) Computer graphics and Rendering
2) Engineering and CAD software
3) Digital plotter and printers
4) Medical imaging
5) Geographical information system.

Conclusion: However, there are scenarios, where other line drawing algorithms might be preferred
1) Bresenham's Algorithm
2) Anti-Aliasing Techiques
3) curved Lines and splines

conclusion:-In this practicle we learn about DDA - digital differential analyzer algorithm and its implementation

Marks out of 10	9
Signature with Date of Completion	

or ellipses.

- It assumes that the line moves from left to right. Adjustments are needed for lines in other directions.

Coding:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <graphics.h>
void main() {
    int x1, x2, y1, y2;
    int gd = DETECT, gm;
    void linebres (int, int, int, int);
    printf ("Enter the two end points = ");
    scanf ("%d %d %d %d", &x1, &x2, &y1, &y2);
    initgraph (&gd, &gm, "");
    cleardevice ();
    linebres (x1, y1, x2, y2);
    getch ();
    line~~bres~~ (x1, y1, x2, y2)
    getch ()
    closegraph ();
}
void linebres (int x1, int x2, int y1, int y2) {
    int dx = abs (x1 - x2), dy = abs (y1 - y2);
    int p, x, y, xend, yend;
    if (dx != 0) {
        p = 2 * dy - dx;
        if (x1 > x2) {
            x = x2;
            y = y2;
            xend = x1;
        } end {
            x = x1
            y = y1
```

```
        xend = dy2
    }
    putpixel(x, y, z);
    for(i=x; i<yend; i++){
        x+=1;
        if(p<0){
            p+=2*dy;
        }
        else{
            p+=2*(dy-dx);
        }
        putpixel(x, y);
    }
} else {
    p = 2*dx-dy;
    if(y1>y2){
        x=x2;
        y=y2;
        yend=y2
    }
    putpixel(x, y1);
    for(i=y; i<yend; i++){
        y+=1;
        if(p<0){
            p+=2*dx;
        }
        else{
            x+=1;
            p+=2*(dx-dy);
        }
        putpixel(x, y, z);
    }
}
```

OUTPUT

Enter the first coordinates : 100 150

Enter the Second coordinates : 150 200

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

Post Practical Questions:

1. **Algorithm Analysis:**
   Compare the Bresenham's Line drawing algorithm with the DDA algorithm. Discuss the advantages and disadvantages of each algorithm in terms of accuracy, efficiency, and ease of implementation.

Ans: Accuracy: The DDA algo doesnot have a high level of precious or accuracy than Boresnham's algo
efficiency: The DDA algo is less efficiented the Boresenham's algo
case of implementation: Boresenham's algo is fuster than DDA algo in line drawing

2. **Integer vs Floating-Point Arithmetic:**
   Bresenham's algorithm uses integer arithmetic for efficiency. Explain the advantages of using integer arithmetic over floating-point arithmetic in graphics algorithms. Discuss any potential challenges and when floating-point arithmetic might be preferred.

Ans: Advantages: Faster calculation
Reduce memory footpaint
No rounding error
Disadvantage: limited precision.
limited functionality.
highly precise calculation

3. **Optimization Techniques:**
   Explore optimization techniques for the Bresenham's algorithm. Discuss how to optimize the algorithm for specific scenarios and provide examples of situations where these optimizations would be beneficial.

Ans: Pore computation.
Symmatry utilization
look up tables.
hardware all atoraction.
GPU
algorithm variations

examples.
real time graphics situations
mobile graphics
jump development

Conclusion: In this practicle we learnt about Boresenham's Line drawing algorithm in c & its implementation

Marks out of 10	
Signature with Date of Completion	

Coding:

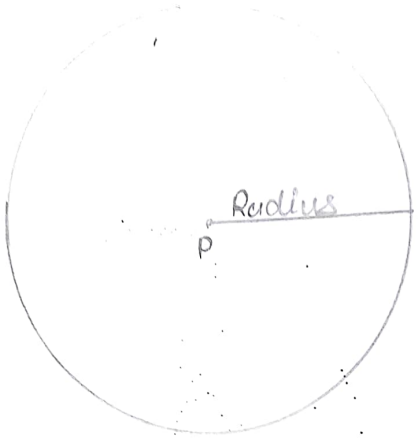
```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void main(){
    int gd= DETECT,gm;
    int x,y,r;
    void drawcircle(int,int,int);
    printf("Enter the mid points and Radius = ");
    scanf("%d %d %d",&x,&y,&r);
    initgraph(&gd,&gm,&x,&y);
    Drawcircle(x,y,r);
    getch();
    closegraph();
}
void drawcircle(int x1, int y1, int r){
    int x=0, y=r, p=1-r;
    void cliplot(int,int,int,int);
    cliplot(x1,y1,x,y);
    while(x<y){
        x++;
        if(p<0){
          p+= 2*x+1;
        }else{
            y--;
          p+= 2*(x-y)+1;
        }
        cliplot(x1,y1,x,y);
    }
}
```

```
void cliplat (int xctor, int yctor, int x, int y) {
    putpixel (xctor+x, yctor+y, 1);
    putpixel (xctor+-x, yctor+y, 1);
    putpixel (xctor+x, yctor-y, 1);
    putpixel (xctor-x, yctor-y, 1);
    putpixel (xctor+y, yctor+x, 1);
    putpixel (xctor-y, yctor+x, 1);
    putpixel (xctor+y, yctor-x, 1)
    putpixel (xctor-y, yctor-x, 1)
    getch();
}
```

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

OUTPUT

Enter center coordinate : 100 100
Enter Radius : 50



Post Practical Questions:

1. Explain the steps involved in the mid-point Circle drawing algorithm. Discuss how the algorithm handles the decision-making process to determine the pixels to be plotted.

Ans: 1) initialization :- (x-c, y-c) its radius is and set the initial point to '(0, r)'.
2) Decision Parameter Calculator : Calculate the decision parameter 'p' using the formula : p = 1-r
3) Plotting points : If p is less than zero
If p is greater than 0 p = p + 2 * x + 1
p = p + 2*(x-y)+1

2. Discuss the limitations of the mid-point Circle drawing algorithm concerning the radius of the circle. How does the algorithm behave for very small or very large radii? Provide insights into potential improvements for handling extreme cases.

Ans: Limitation for small radii : When the radius very small algorithm may not produce accurate radius this algorithm is efficient for scan conversion. for creating geometric curves on raster display to generate curves on raster display.

3. Identify and discuss practical applications where drawing circles is essential. Explain why the mid-point Circle drawing algorithm is suitable for these applications and when other circle drawing algorithms might be preferred.

Ans: In computer graphics this algorithm is used to determine the points needed for rastering a circle, mid point circle algorithm plots on the first gradient two regions

Conclusion: In this practicle we learnt develop the mid point circle drawing algorithm using c language

Marks out of 10	
Signature with Date of Completion	

```
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 - 1, y, fill_color, boundary_color);
    boundaryFill4(x, y + 1, fill_color, boundary_color);
    boundaryFill4(x, y - 1, fill_color, boundary_color);
  }
}
```

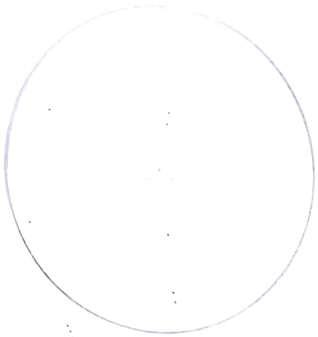
Coding:

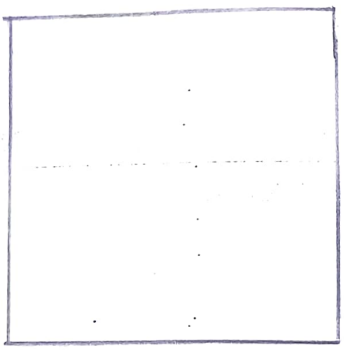
```
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
void boundaryfill(int x, int y, int f_colour, int b_colour)
{
  if(getpixel(x,y) != b_colour && getpixel(x,y) !=
                                        f_colour)
  {
    putpixel(x, y, f_colour);
    boundaryfill(x+1, y, f_colour, b_colour);
    boundaryfill(x, y+1, f_colour, b_colour);
    boundaryfill(x, y-1, f_colour, b_colour);
    boundaryfill(x-1, y, f_colour, b_colour);
  }
}

// getpixel(x,y) gives the colour of the specified pixel
```

```
int main() {
    int gm, gd= DETECT, radius;
    int x,y;
    printf("Enter x and y position for circle\n");
    scanf("%d %d",&x&y);
    printf("Enter the radius or circle\n");
    scanf("%d" &radius
    initgraph(&gd, &gm, "C:\\turboc3\\bgi");
    circle(x, y, radius);
    boundaryfill(x,y,4,15);
    delay(5000);
    closegraph();
    return 0;
}
```

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

OUTPUT





Post Practical Questions:

1. Compare and contrast the 8-connected and 4-connected boundary fill algorithms. Discuss scenarios where one algorithm might be more suitable than the other and provide examples.

Ans: In 4 connect method the pixel can have at maximum 4 neighbours that positioned at the proper left, above & below the present pixel. In 8 connected method at can have eight ~~neighbouring~~ boundary position are checked against four diagonal pixels

2. Explain how the boundary fill algorithms handle interior points and the impact on the filling process. Discuss any potential challenges and solutions for dealing with interior points efficiently.

Ans: In takes an interior points (x, y) a fill ~~fill~~ color and a boundary 29 color as the input, the algorithm step by checking the color at (x, y) If 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)

3. Discuss limitations of the boundary fill algorithms in handling complex shapes and suggest improvements or alternative strategies for addressing these limitations.

Ans: Boundary fill flood fill algorithm like if an inside pixel is in some other color then the fill terminates and the polygon remains unfilled. It does not work merge polygons

Conclusion:

In this practical we learnt to develop the 8-connected and 4-connected boundary fill algorithm using clanguage

Marks out of 10	
Signature with Date of Completion	