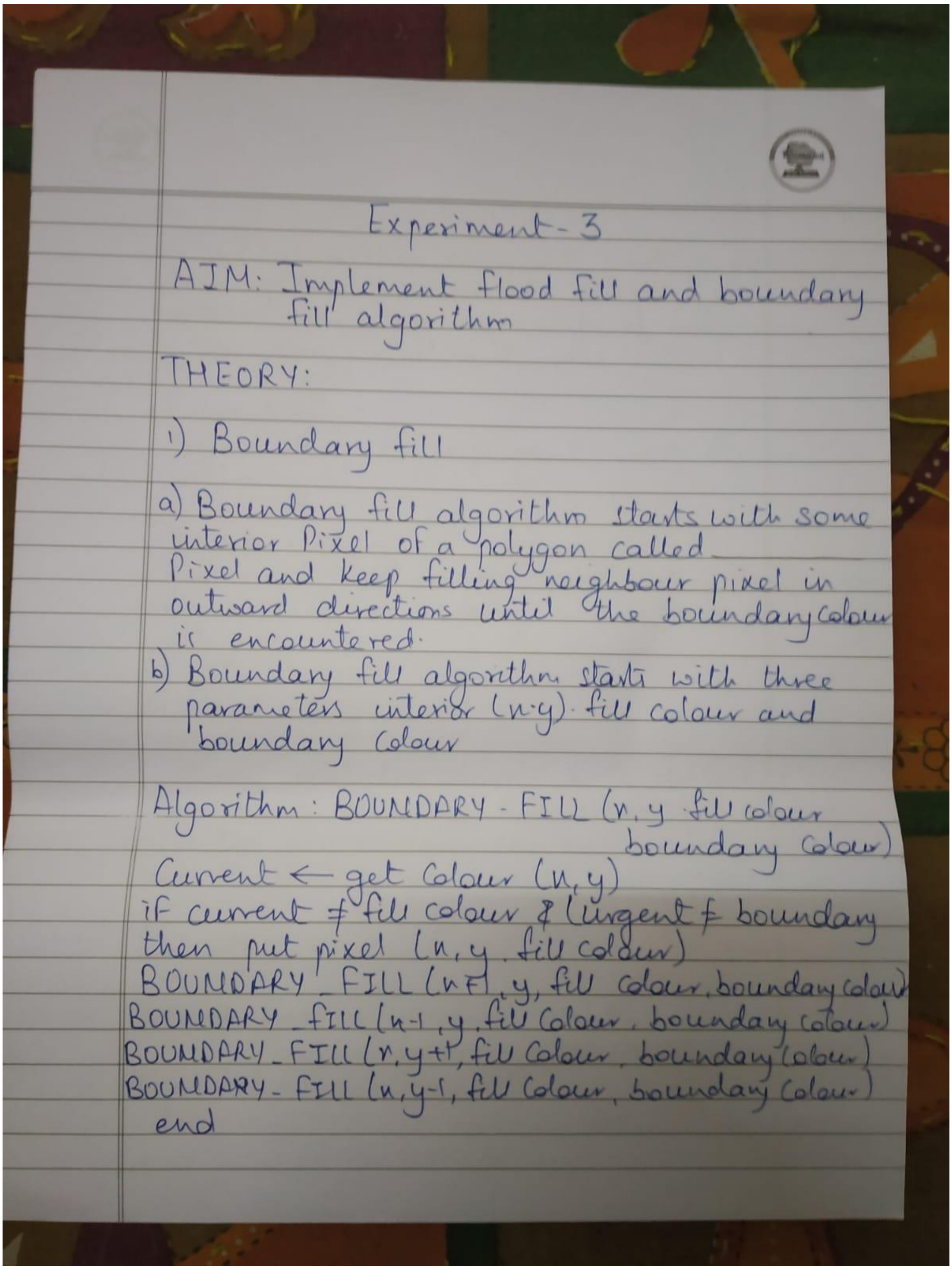
Assignment performed by Gaurav Amarnani from DSE CMPN.
CG LAB 3 - Implement Flood Fill and Boundary Fill Algorithm.



Experiment - 3

AIM: Implement flood fill and boundary fill algorithm

THEORY:

1) Boundary fill

a) Boundary fill algorithm starts with some interior Pixel of a polygon called Pixel and keep filling neighbour pixel in outward directions until the boundary colour is encountered.

b) Boundary fill algorithm starts with three parameters interior (x.y). fill colour and boundary colour

Algorithm: BOUNDARY - FILL (x, y fill colour boundary colour)

```
Current ← get Colour (x, y)
if current ≠ fill colour & (urgent ≠ boundary
then put pixel (x, y. fill colour)
BOUNDARY_FILL (x+1, y, fill colour, boundary colour)
BOUNDARY_FILL (x-1, y. fill Colour, boundary colour)
BOUNDARY_FILL (x, y+1, fill Colour, boundary colour)
BOUNDARY_FILL (x, y-1, fill Colour, boundary Colour)
end
```

This algorithm may not work properly if some pixels are already filled, because algo returns when the current pixel is either boundary pixel or it was filled

Advantages:
Simple easy to implement, work for any type of polygon

Disadvantages:
Requires extensives stacking due to recursion requires more memory. not suitable for large polygon.

2) Flood fill

Many realistic objects have different coloured boundaries boundary fill algorithm cannot fill the polygon with the multi coloured boundary. flood fill can handle this issue

Algorithm

```
flood-fill (n, y. colour, old Colour)

Current ← get Color (x, y)
if current ≠ old Colour then
put pixel (n, y, fill Colour)
```

```
flood-fill (n+1, y-fill Color, old Color)
flood_fill (n, y+1, fill Color, old Color)
flood-fill (n, y-1, fill Color, old Color)
end
```

Advantages:

Simple easy to understand boundary fill algorithm cannot handle the object with multi-Color boundary, whereas flood fill cap

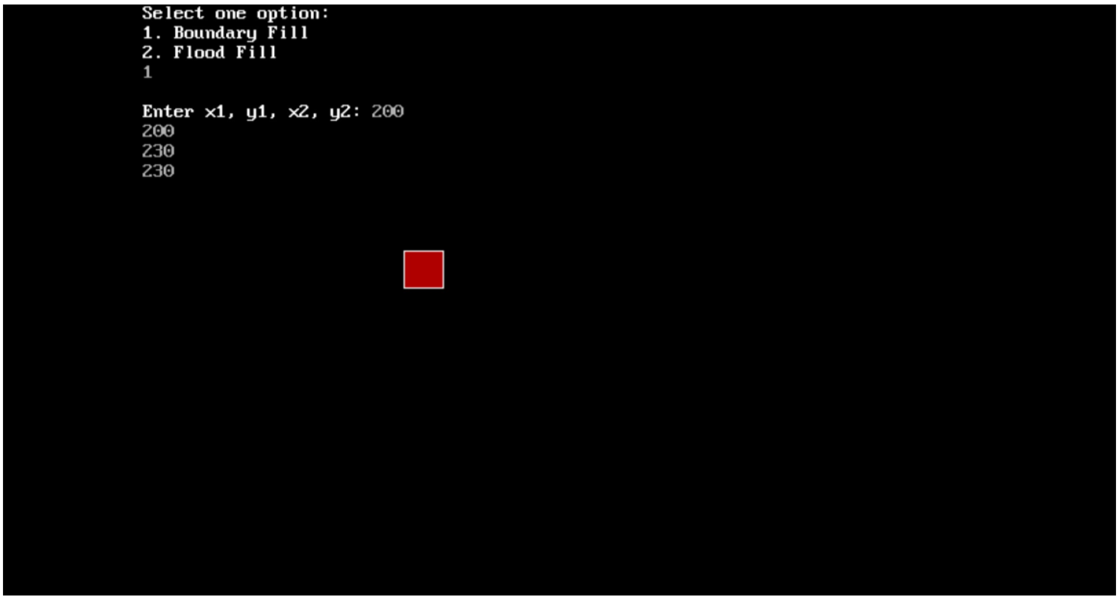
Disadvantages

- Requires extensive stacking
- slow in nature
- Not suitable for large polygon

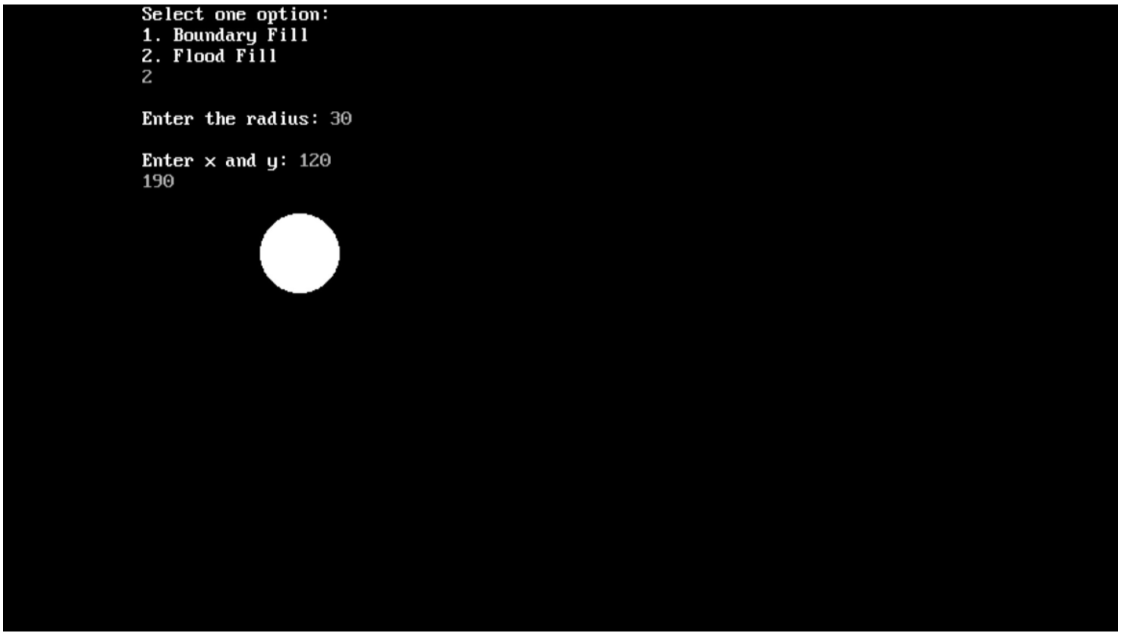
Program:

```
#include <graphics.h>

//Performed by Gaurav Amarnani DSE CMPN.

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);
}
}

void flood(int x, int y, int oldcol, int newcol) {

if(getpixel(x,y) == oldcol) {
putpixel(x,y,newcol);
delay(1);
flood(x+1,y,oldcol,newcol);
flood(x,y+1,oldcol,newcol);
flood(x-1,y,oldcol,newcol);
flood(x,y-1,oldcol,newcol);
}
}

void main() {
int gd = DETECT, gm;
int x,y,radius,x1,y1,x2,y2,ch;
initgraph(&gd, &gm, "c:\\turboc3\\bgi");
printf("Select one option:\n1. Boundary Fill\n2. Flood Fill\n");
scanf("%d",&ch);
switch(ch) {
case 1:
printf("\nEnter x1, y1, x2, y2: ");
scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
rectangle(x1, y1, x2, y2);
boundaryFill4((x1+1), (y1+1), 4, 15);
break;

case 2:
printf("\nEnter the radius: ");
scanf("%d",&radius);
printf("\nEnter x and y: ");
scanf("%d %d",&x,&y);
circle(x,y,radius);
flood(x,y,0,15);
break;

default:
printf("Enter valid option.");
break;
}
getch();
closegraph();
}
```

Output:

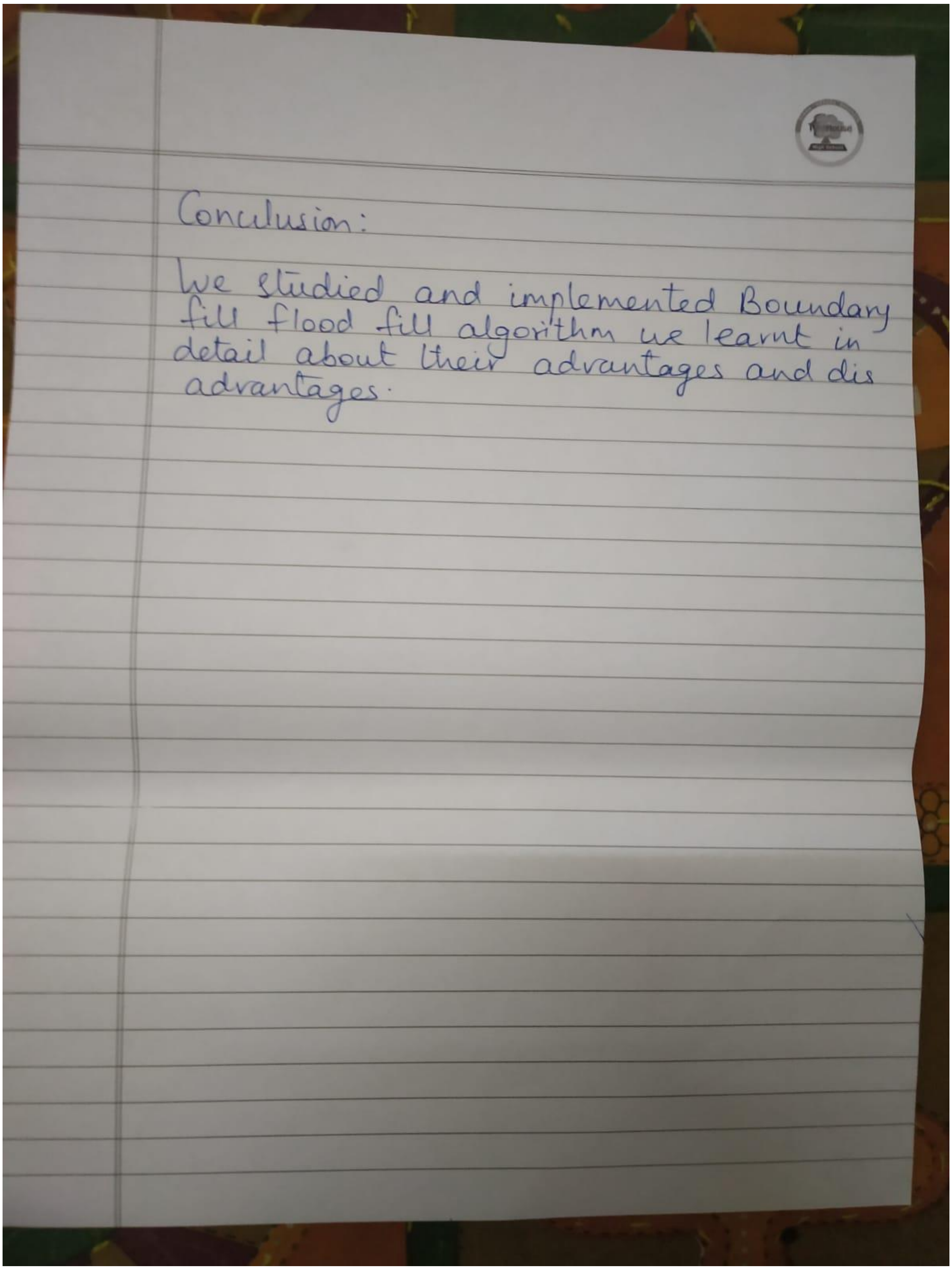


```
Select one option:
1. Boundary Fill
2. Flood Fill
1

Enter x1, y1, x2, y2: 200
200
230
230
```



```
Select one option:
1. Boundary Fill
2. Flood Fill
2

Enter the radius: 30

Enter x and y: 120
190
```

Conclusion:



Conclusion:

We studied and implemented Boundary fill flood fill algorithm we learnt in detail about their advantages and dis advantages.