Name: Sunny Solanky
Course: BCA 6C
Rollno: 36 5112115 19
Sub: Computer Graphics. PBC 602

Ans1.
```
#Include <stdio.h>
# Include < graphics. h>
int main C)
{
 int gd= DETECT, gm, x0, y0, y1, x1, dx
 dy, p, x, y;
 printf(" Co-ordinates of first point:");
 printf("\n Enter the value of x1:");
 scanf("%d", &x0);
 printf(" Enter the value of y1: ");
 scanf("d", &y0);
 printf(" Co-ordinates of second point:");
 printf("\n Enter the value of x2:");
 scanf("%.d", &x1 );

 printf(" Enter the value of x2: ");
 scanf("%.d", &x1);
 printf("\n Enter the value of y2:");
 scanf("%.d", &y1);
 initgraph (&gd, &gm, "");
 dx= x1-x0;
```

```
dy = y1 - y0;
x = x0;
y = y0;
p = 2*dy - dx;
while (x<x1)
{
  if (p>=0)
  {
    putpixel (x,y,4);
    y = y+1;
    p = p+ 2*dy - 2*dx
  }
    else
    {
      putpixel(x,y,4);
      p = p+2*dy;
    }
    x=x+1;
  }
    getch();
    return 0;
}
```

Bresenham's Line Algorithm:

step 1: Start Algorithm

step 2: Declare variable

$x_1, x_2, y_1, y_2, p1, p2, dx, dy$

step 3: Enter value of x_1, y_1, x_2, y_2 where $x1, y1$ are coordinates of starting point and $x_2, y2$ are coordinates of Ending point.

step 4: Calculate $dx = x_2 - x_1$

$dy = y_2 - y_1$

$p1 = 2 * dy$

$p2 = 2 * (dy - dx)$

$d = p1 - dx$

step 5: Consider (x, y) as starting point and xend as maximum possible value of x

If $dx < 0$

then $x = x_2$

$y = y_2$

$xend = x1$

if $dx > 0$

then $x = x1$

$y = y_1$

$xend = x_2$

step 6: Generate point at (x, y)

step 7: check if whole line is
if $x >= x_{end}$
stop

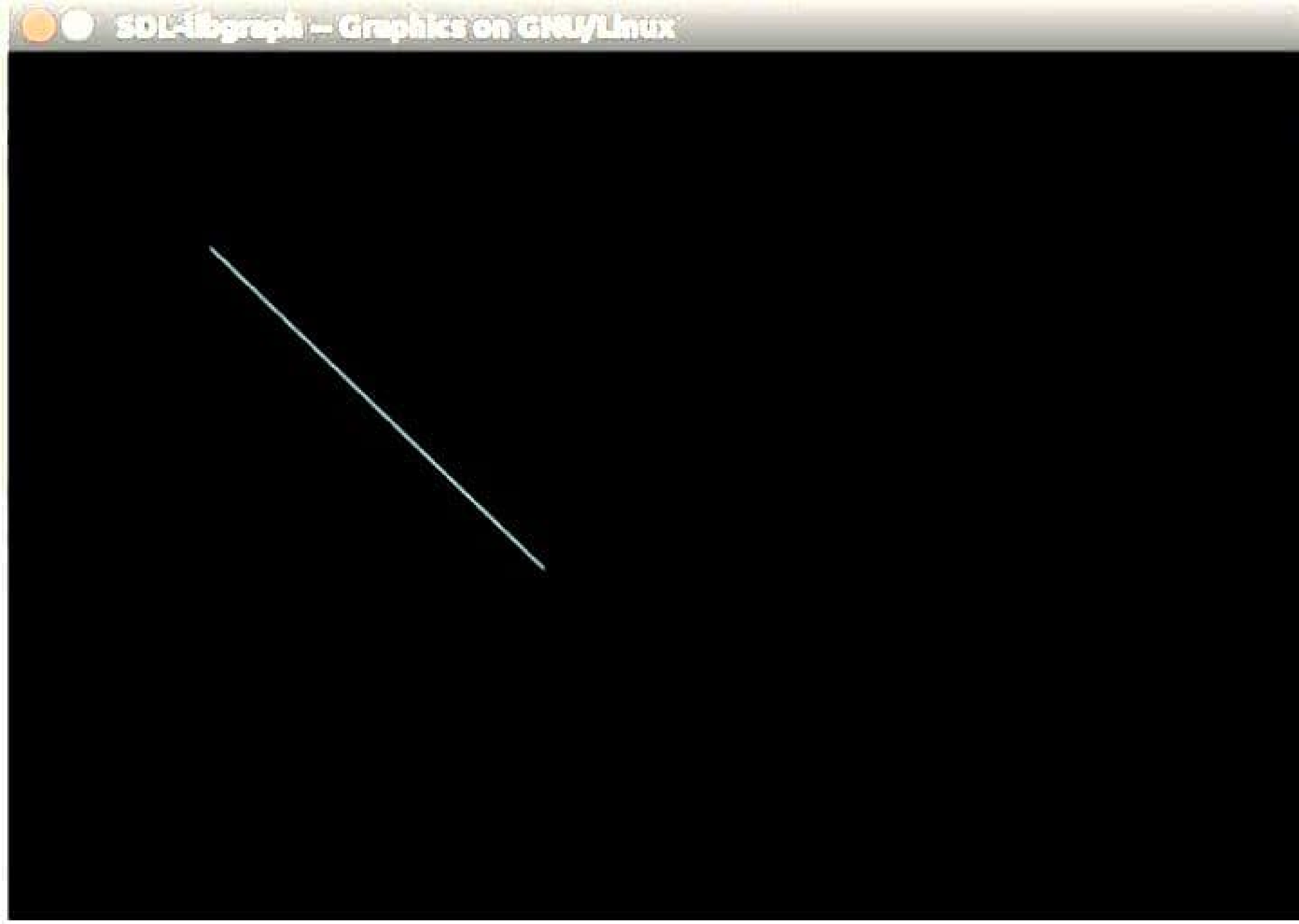
step 8: next pixel
if $d < 0$
then $d = d + p_1$
if $d \geq 0$
then $d = d + p_2$
inor $y = y + 1$

step 9: increament $x = x + 1$

step 10: Draw point of latest (x, y) coordinates

step 11: Go to step 7

step 12: stop.


SDL_gfxprim – Graphics on GNU/Linux

Ans2.

```
#include <stdio.h>
#include <graphics.h>
void drawcircle (int x0, int y0, int radius)
{
  int x = radius;
  int y = 0;
  int err = 0;
  while (x >= y)
  {
    putpixel (x0+x, y0+y, 7);
    putpixel (x0+y, y0+x, 7);
    putpixel (x0+y, y0+x, 7);
    putpixel (x0-x, y0+x, 7);
    putpixel (x0-x, y0-y, 7);
    putpixel (x0, y, y0-x, 7);
    putpixel (x0+y, y0+x, 7);
    putpixel (x0+x, y0-y, 7);
  if (err <= 0)
  {
    y++;
    err += 2*dy + 1;
  }
  if (err > 0)
  {
    x--;
    err -= 2*x + 1;
```

```
}
}
int main()
{
    int gd = DETECT, gm, error, x,
    y, r;
    printf("Enter radius of circle: ");
    scanf("%d", &r);
    printf("Enter co-ordinates of center
            (x and y): ");
    scanf("%d%d", &x, &y);
    initgraph(&gd, &gm, "");
    drawcircle(x, y, r);
    delay(99999999);
    return 0;
}
```

Mid point Circle

Step 1: Start

Step 2: Put $x = 0$, $y = r$ in equation 2 we have $p = 1 - r$

Step 3: Repeat steps while $x \leq y$

plot (x, y)

if $(p < 0)$

Then set $p = p + 2x + 3$

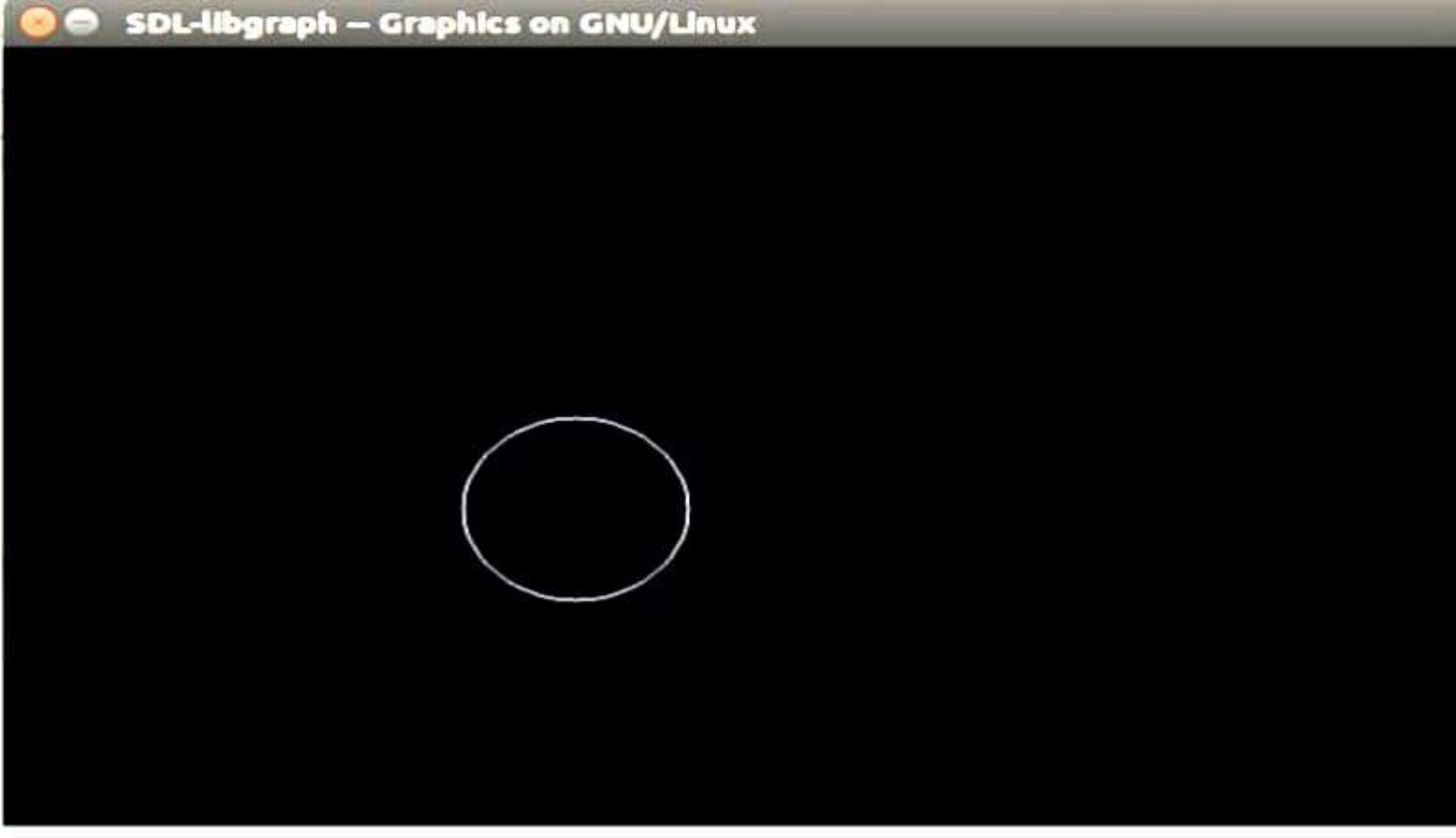
else

$p = p + 2(x+y) + 5$

& $y = y - 1$ (end if)

$x = x + 1$ (end loop)

Step 4 Step 4: ~~En~~ Stop.


SDL-libgraph – Graphics on GNU/Linux

Ans3. Alam

```
#include <graphics.h>
#include <stdio.h>
void boundary_fill (int x, int y, int
  fill_color, int bound_color)

{
 if (getpixel (x,y) != fill_color &&
   getpixel (x,y) != bound_color)
  {
   putpixel (x, y, fill_color);
   delay (1);
   boundary_fill (x+1, y, fill_color,
     bound_color); }
   boundary_fill (x, y-1, fill_color,
     bound_color);
   boundary_fill (x-1, y, fillcolor,
       bound_color);
   boundary_fill (x, y+1, fill_color,
       bound_color);
   boundary_fill (x-1, y-1, fill_color,
       bound_color);
   boundary_fill (x+1, y-1, fill_color,
     bound_color); boundary_fill
    (x-1, y+1; fill_color, bound_
```

```
color);
boundary-fill (x+1, y+1, fill_color, bound_
                Color);
}
}
int main()
{
int gd = DETECT, gm;
 initgraph (&gd, &gm, "");
   line (100, 100, 250, 100);
   line (250, 100, 250, 250);
    line (250, 250, 400, 250);
     line (400, 250, 400, 400);
     line (248, 400, 400, 400);
      line (248, 250, 248, 400);
     line (100, 100, 100, 250);
      boundary fill (150, 150, RED, WHITE);
    getch()
    closegraph();
    return 0;
   }
```

Algorithm for Bamdary fill 8 connecte
Algorithm.

Step 1: Create a function named as boundary with 8 parameters (x, y, fill_color, bound_color).

Step 2: Call it recursively untill the boundary pixels are reached.

Step 3: Stop.