Practical 1 :

Write C++/JAVA program for line drawing using DDA or Bresenhams algorithm with patterns such as solid, dotted, dashed, dash dot and thick of.

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

void bresenham(int x1, int y1, int x2, int y2) {
   int dx, dy, p, x, y;

   dx=x2-x1;
   dy=y2-y1;

   x=x1;
   y=y1;

   p=2*dy-dx;

   while(x<x2) {
      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, x1, y1, x2, y2;
   initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");

   cout<<"Enter co-ordinates of first point: ";
   cin>>x1>>y1;

   cout<<"Enter co-ordinates of second point: ";
   cin>>x2>>y2;
   bresenham(x1, y1, x2, y2);

   return 0;
}
```

Practical 2

Write C++/JAVA program to draw circle using Bresenham's algorithm. Inherit pixel class of.

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

class Pixel {
public:
   void put_pixel(int x, int y, int color) {
      putpixel(x, y, color);
   }
};

class Circle : public Pixel {
public:
   void draw_circle(int xc, int yc, int r) {
      int x = 0, y = r;
      int d = 3 - 2 * r;
      draw_symmetry(xc, yc, x, y);
      while (y >= x) {
         x++;
         if (d > 0) {
            y--;
            d = d + 4 * (x - y) + 10;
         } else {
            d = d + 4 * x + 6;
         }
         draw_symmetry(xc, yc, x, y);
      }
   }
```

```
private:
    void draw_symmetry(int xc, int yc, int x, int y) {
        put_pixel(xc+x, yc+y, 7);
        put_pixel(xc-x, yc+y, 7);
        put_pixel(xc+x, yc-y, 7);
        put_pixel(xc-x, yc-y, 7);
        put_pixel(xc+y, yc+x, 7);
        put_pixel(xc-y, yc+x, 7);
        put_pixel(xc+y, yc-x, 7);
        put_pixel(xc-y, yc-x, 7);
    }
};

int main() {
    int gdriver = DETECT, gmode, error, xc, yc, r;
    initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");

    cout << "Enter center of the circle: ";
    cin >> xc >> yc;
    cout << "Enter radius of the circle: ";
    cin >> r;

    Circle c;
    c.draw_circle(xc, yc, r);

    return 0;
}
```

Practical 3

Write C++/Java program to draw circle using Bresenham‘s algorithm. Inherit pixel class

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

class Pixel {
public:
   int x, y;
   Pixel(int x, int y) : x(x), y(y) {}
};

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 bresenhamCircle(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);
    }
}

int main()
{
    int xc = 50, yc = 50, r = 30;
    int gd = DETECT, gm;
    initgraph(&gd, &gm, NULL);
    bresenhamCircle(xc, yc, r);
    getch();
    closegraph();
    return 0;
}
```

Practical 4

Write C++ program to draw a concave polygon and fill it with desired color using scan fill algorithm. Apply the concept of inheritance.

```
#include <graphics.h>
#include <iostream.h>
#include <conio.h>
class Shape {
protected:
   int x, y;

public:
   Shape(int x, int y) {
      this->x = x;
      this->y = y;
   }

   virtual void draw() = 0;
};

class Polygon : public Shape {
private:
   int numPoints;
   int* points;

public:
   Polygon(int x, int y, int numPoints, int* points) : Shape(x, y) {
      this->numPoints = numPoints;
      this->points = points;
   }
```

```
    void draw() {
        int gd = DETECT, gm;
        initgraph(&gd, &gm, "");

        setcolor(15); // Set color to white
        fillpoly(numPoints, points);

        getch();
        closegraph();
    }
};

int main() {
    int x = 100, y = 100;
    int numPoints = 6;
    int points[] = { 50, 50, 150, 50, 200, 100, 150, 150, 100, 150, 50, 100 };

    Polygon polygon(x, y, numPoints, points);
    polygon.draw();

    return 0;
}
```

Practical 5

Write C++ program to implement Cohen Southerland line clipping algorithm.

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#define TOP 8
#define BOTTOM 4
#define RIGHT 2
#define LEFT 1
int getcode(int x,int y, int xl,int yl,int xh,int yh)
{
   int code = 0;
   if(y > yh)
      code |= TOP;
   else if(y < yl)
      code |= BOTTOM;
   if(x > xh)
      code |= RIGHT;
   else if(x < xl)
      code |= LEFT;
   return code;
}
void cohen_sutherland_lineclip_and_draw(int x1,int y1,int x2,int y2,int xl,int yl,int xh,int yh)
{
   int code1, code2, codeout;
   int accept = 0, done = 0;
   code1 = getcode(x1,y1,xl,yl,xh,yh);
   code2 = getcode(x2,y2,xl,yl,xh,yh);
   do
   {
      if(!(code1 | code2))
```

```
{
   accept = 1;
   done = 1;
}
else if(code1 & code2)
   done = 1;
else
{
   int x,y;
   codeout = code1 ? code1 : code2;
   if(codeout & TOP)
   {
      x = x1 + (x2 - x1) * (yh - y1) / (y2 - y1);
      y = yh;
   }
   else if(codeout & BOTTOM)
   {
      x = x1 + (x2 - x1) * (yl - y1) / (y2 - y1);
      y = yl;
   }
   else if(codeout & RIGHT)
   {
      y = y1 + (y2 - y1) * (xh - x1) / (x2 - x1);
      x = xh;
   }
   else
   {
      y = y1 + (y2 - y1) * (xl - x1) / (x2 - x1);
      x = xl;
   }
   if(codeout == code1)
```

```
            {
               x1 = x;
               y1 = y;
               code1 = getcode(x1,y1,xl,yl,xh,yh);
            }
            else
            {
               x2 = x;
               y2 = y;
               code2 = getcode(x2,y2,xl,yl,xh,yh);
            }
         }
      }while(done == 0);
      if(accept)
         line(x1,y1,x2,y2);
}
void main()
{
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TC\\BGI");
   int x1, y1, x2, y2;
   cout<<"Enter coordinates of the line: ";
   cin>>x1>>y1>>x2>>y2;
   int xl, yl, xh, yh;
   cout<<"Enter coordinates of the clipping window: ";
   cin>>xl>>yl>>xh>>yh;
   rectangle(xl, yl, xh, yh);
   cohen_sutherland_lineclip_and_draw(x1,y1,x2,y2,xl,yl,xh,yh);
   getch();
   closegraph();
}
```

Practical 6

Write C++ program to draw a given pattern. Use DDA line and Bresenham's circle drawing algorithm. Apply the concept of encapsulation.

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h> // Include this library to use the abs function
class DDA {
public:
   void line(int X0, int Y0, int X1, int Y1) {
      int dx = X1 - X0;
      int dy = Y1 - Y0;
      int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);
      float Xinc = dx / (float) steps;
      float Yinc = dy / (float) steps;
      float X = X0;
      float Y = Y0;
      for (int i = 0; i <= steps; i++) {
         putpixel (X,Y,WHITE);
         X += Xinc;
         Y += Yinc;
      }
   }
};
class Bresenham {
public:
   void circle(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);
         } }
private:
   void drawCircle(int xc, int yc, int x, int y) {
      putpixel(xc+x, yc+y, WHITE);
      putpixel(xc-x, yc+y, WHITE);
      putpixel(xc+x, yc-y, WHITE);
      putpixel(xc-x, yc-y, WHITE);
      putpixel(xc+y, yc+x, WHITE);
      putpixel(xc-y, yc+x, WHITE);
      putpixel(xc+y, yc-x, WHITE);
      putpixel(xc-y, yc-x, WHITE);
   }
};
void main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TC\\BGI");
   DDA dda;
   Bresenham bresenham;
   dda.line(150, 150, 450, 150);
   bresenham.circle(300, 300, 100);
   getch();
   closegraph();
}
```

Practical 7

a) Write C++ program to draw 2-D object and perform following basic transformations: Scaling, Translation, Rotation. Apply the concept of operator overloading.

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
class Transform {
   int x, y;
public:
   Transform() : x(0), y(0) {}
   Transform(int x, int y) : x(x), y(y) {}

   // Overload the * operator for scaling
   Transform operator*(int scale) {
      return Transform(x*scale, y*scale);
   }

   // Overload the + operator for translation
   Transform operator+(const Transform& t) {
      return Transform(x+t.x, y+t.y);
   }

   // Overload the - operator for rotation
   Transform operator-(double angle) {
      double rad = angle * 3.14159 / 180;
      return Transform(x*cos(rad) - y*sin(rad), x*sin(rad) + y*cos(rad));
   }
```

```
    void draw() {
        circle(x, y, 10);
    }
};

void main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TC\\BGI");

    Transform t1(100, 100);
    t1.draw();

    // Perform scaling
    Transform t2 = t1 * 2;
    t2.draw();

    // Perform translation
    Transform t3 = t2 + Transform(50, 50);
    t3.draw();

    // Perform rotation
    Transform t4 = t3 - 45;
    t4.draw();

    getch();
    closegraph();
}
```

Practical 8

a) Write C++ program to generate snowflake using concept of fractals.

```
#include <iostream.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>

void snowflake(int x1, int y1, int x2, int y2, int depth) {
   if (depth == 0) {
      line(x1, y1, x2, y2);
   } else {
      int dx = x2 - x1;
      int dy = y2 - y1;
      int x3 = x1 + dx / 3;
      int y3 = y1 + dy / 3;
      int x4 = x3 + dx / 2 - dy / 2;
      int y4 = y3 + dy / 2 + dx / 2;
      int x5 = x1 + dx * 2 / 3;
      int y5 = y1 + dy * 2 / 3;

      snowflake(x1, y1, x3, y3, depth - 1);
      snowflake(x3, y3, x4, y4, depth - 1);
      snowflake(x4, y4, x5, y5, depth - 1);
      snowflake(x5, y5, x2, y2, depth - 1);
   }
}

void main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TC\\BGI");
```

```
    int depth = 4;
    int size = 200;
    int x1 = getmaxx() / 2 - size / 2;
    int y1 = getmaxy() / 2 + size * sqrt(3) / 6;
    int x2 = getmaxx() / 2 + size / 2;
    int y2 = y1;
    int x3 = getmaxx() / 2;
    int y3 = getmaxy() / 2 - size * sqrt(3) / 3;

    snowflake(x1, y1, x2, y2, depth);
    snowflake(x2, y2, x3, y3, depth);
    snowflake(x3, y3, x1, y1, depth);

    getch();
    closegraph();
}
```

Practical 9

Write OpenGL program to draw Sun Rise and Sunset.

```
#include <graphics.h>
#include <conio.h>
#include <dos.h>
#include <math.h>

void drawSun() {
   setcolor(YELLOW);
   setfillstyle(SOLID_FILL, YELLOW);

   int x = getmaxx() / 2;
   int y = getmaxy() / 2;
   int radius = 50;

   fillellipse(x, y, radius, radius);
}

void drawSky() {
   setcolor(LIGHTBLUE);
   setfillstyle(SOLID_FILL, LIGHTBLUE);

   bar(0, 0, getmaxx(), getmaxy());
}

int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

   setbkcolor(WHITE);
```

```
    while (!kbhit()) {
        drawSky();
        drawSun();

        delay(100); // Introduce a delay for animation
        cleardevice(); // Clear the screen for the next frame
    }

    getch();
    closegraph();
    return 0;
}
```

Practical 10

a) Write a C++ program to control a ball using arrow keys. Apply the concept of polymorphism.

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

class Shape {
public:
   virtual void draw() = 0;
   virtual void move() = 0;
};

class Ball : public Shape {
   int x, y, radius;

public:
   Ball(int startX, int startY, int startRadius) : x(startX), y(startY), radius(startRadius) {}

   void draw() override {
      setcolor(RED);
      setfillstyle(SOLID_FILL, RED);
      fillellipse(x, y, radius, radius);
   }

   void move() override {
      if (kbhit()) {
         char ch = getch();
         switch (ch) {
            case 'a':
               x -= 5;
```

```
                break;
            case 'd':
                x += 5;
                break;
            case 'w':
                y -= 5;
                break;
            case 's':
                y += 5;
                break;
        }
    }
}
};

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

    setbkcolor(WHITE);

    Ball ball(100, 100, 20);

    while (!kbhit()) {
        ball.move();
        cleardevice();
        ball.draw();
        delay(50);
    }
    closegraph();
    return 0;
```

}