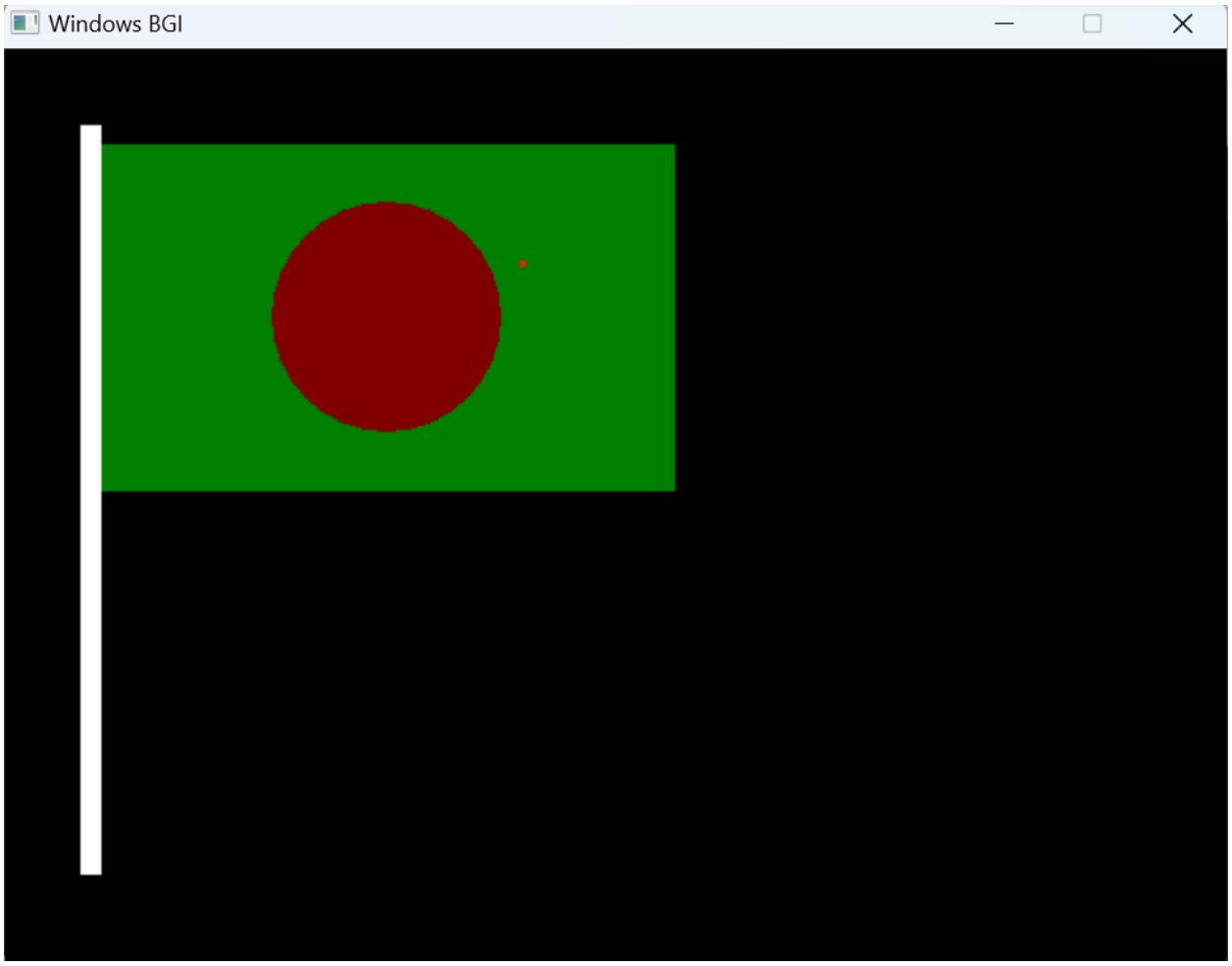
Experiment 1 : Draw the National Flag of Bangladesh

```
#include<bits/stdc++.h>
#include<graphics.h>

using namespace std;

int main()
{
   int gd=DETECT, gm;
   initgraph(&gd, &gm, "");

   setcolor(GREEN);
   rectangle(50,50,350,230);
   setfillstyle(SOLID_FILL, GREEN);
   floodfill(51,51, GREEN);

   setcolor(RED);
   circle(200,140,60);
   setfillstyle(SOLID_FILL, RED);
   floodfill(201,141, RED);

   setcolor(WHITE);
   rectangle(40,40,50,430);
   setfillstyle(SOLID_FILL, WHITE);
   floodfill(41,41, WHITE);

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

Output :



Experiment 2 : Simulate two dimensional geometric Translation, Rotation & Scaling

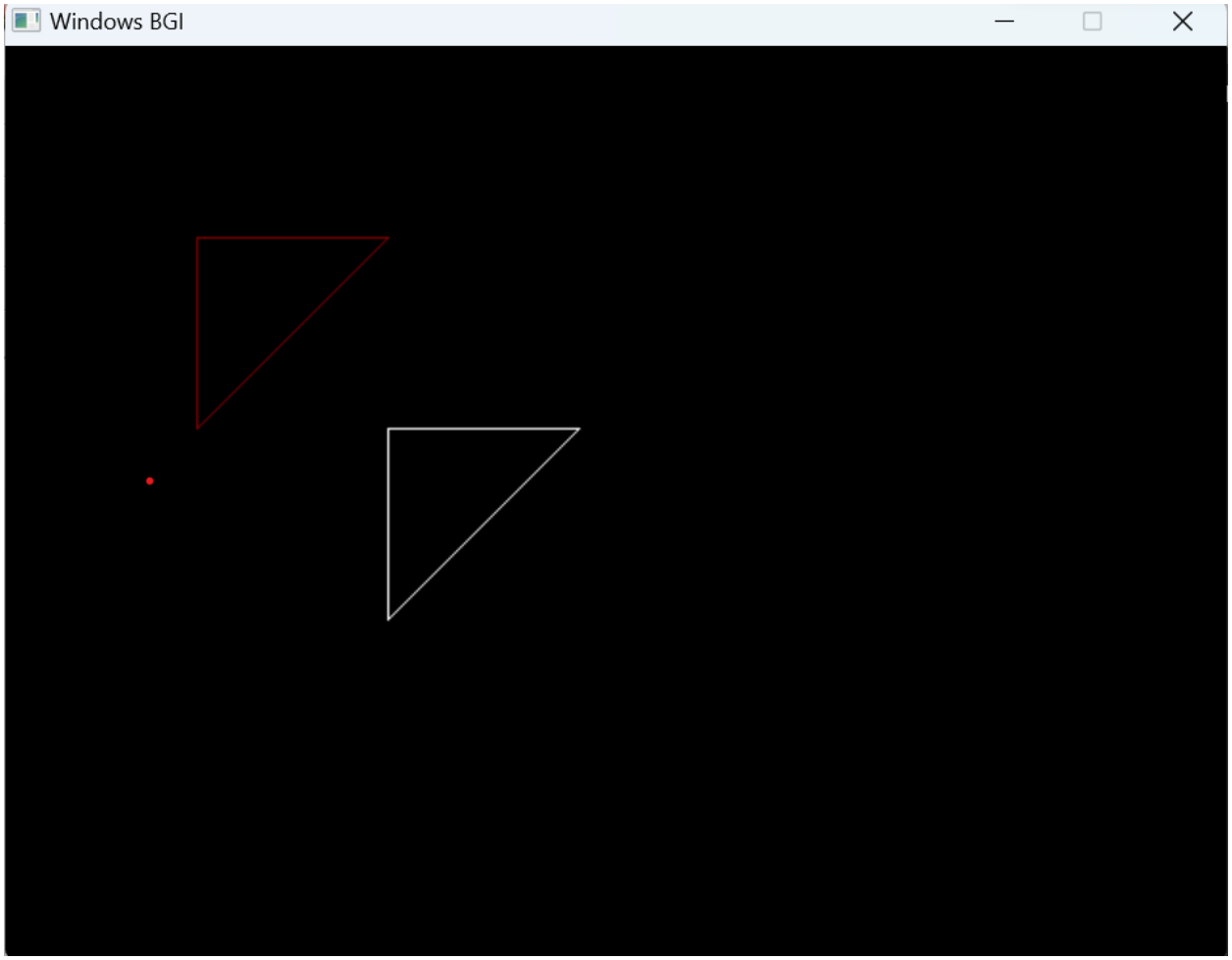
```
#include<bits/stdc++.h>
#include<graphics.h>

#define Sin(x) sin(x * acos(-1.0)/180)
#define Cos(x) cos(x * asin(-1.0)/180)

int point, x[10], y[10];
```

```
int tx, ty;  // translation factors
int sx, sy;  // scaling factors
int angle;   // anti clockwise rotation angle
int xpivot, ypivot; // pivot point of coordinates

using namespace std;

void drawpoly()
{
  for(int i=0;i<point;i++)
  {
    line(x[i], y[i], x[(i+1)%point], y[(i+1)%point]);
  }
}

void translation()
{
  for(int i=0;i<point;i++)
  {
    x[i] += tx;
    y[i] += ty;
  }
}

void scaling()
{
  for(int i=0;i<point;i++)
  {
    x[i] *= sx;
    y[i] *= sy;
  }
}

void rotation()
```

```
{
  for(int i=0;i<point;i++)
  {
    int xshift = x[i] - xpivot;
    int yshift = y[i] - yshift;

    x[i] = xpivot + (xshift * Cos(angle) - yshift * Sin(angle));
    y[i] = ypivot + (xshift * Sin(angle) + yshift * Cos(angle));
  }
}

int main()
{
  int gd=DETECT, gm;
  initgraph(&gd, &gm, "");

  cout << "Enter the number of points : " ;
  cin >> point;

  cout << "Enter the coordinates points : " << endl;
  for(int i=0;i<point;i++)
  {
    cin >> x[i] >> y[i];
  }
  setcolor(RED);
  drawpoly();

  char ch;
  cout << "Which Translation ? Transformation -> T, Scaling -> S,
Rotation -> R : ";
  cin >> ch;

  if(ch == 'T')
  {
```

```
    cout << "Enter the translation factor : ";
    cin >> tx >> ty;
    translation();
    setcolor(WHITE);
    drawpoly();
  }
```

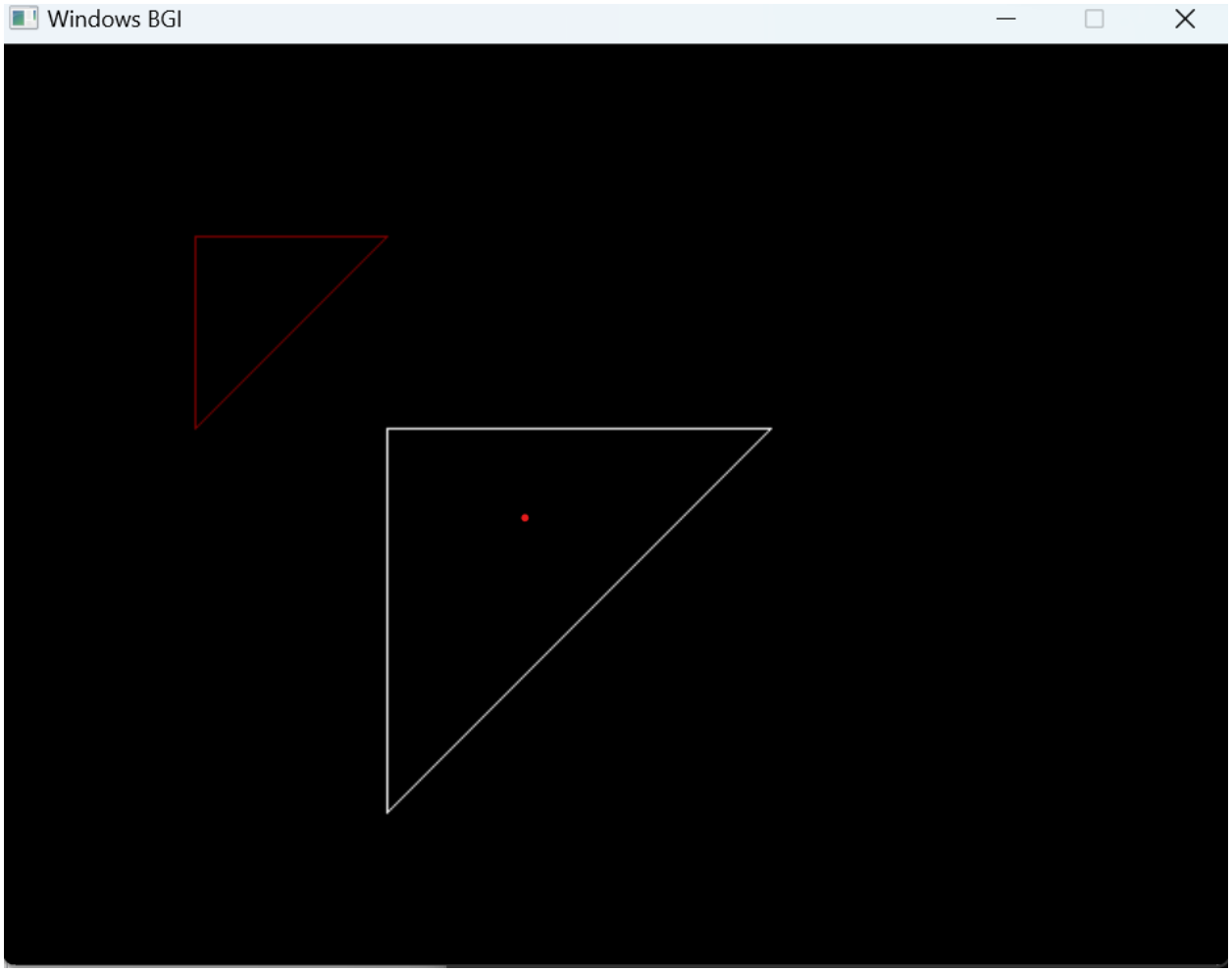
output:




```
  else if(ch == 'S')
  {
    cout << "Enter the scaling factor : ";
    cin >> sx >> sy;
```

```
    scaling();
    setcolor(WHITE);
    drawpoly();
  }
```

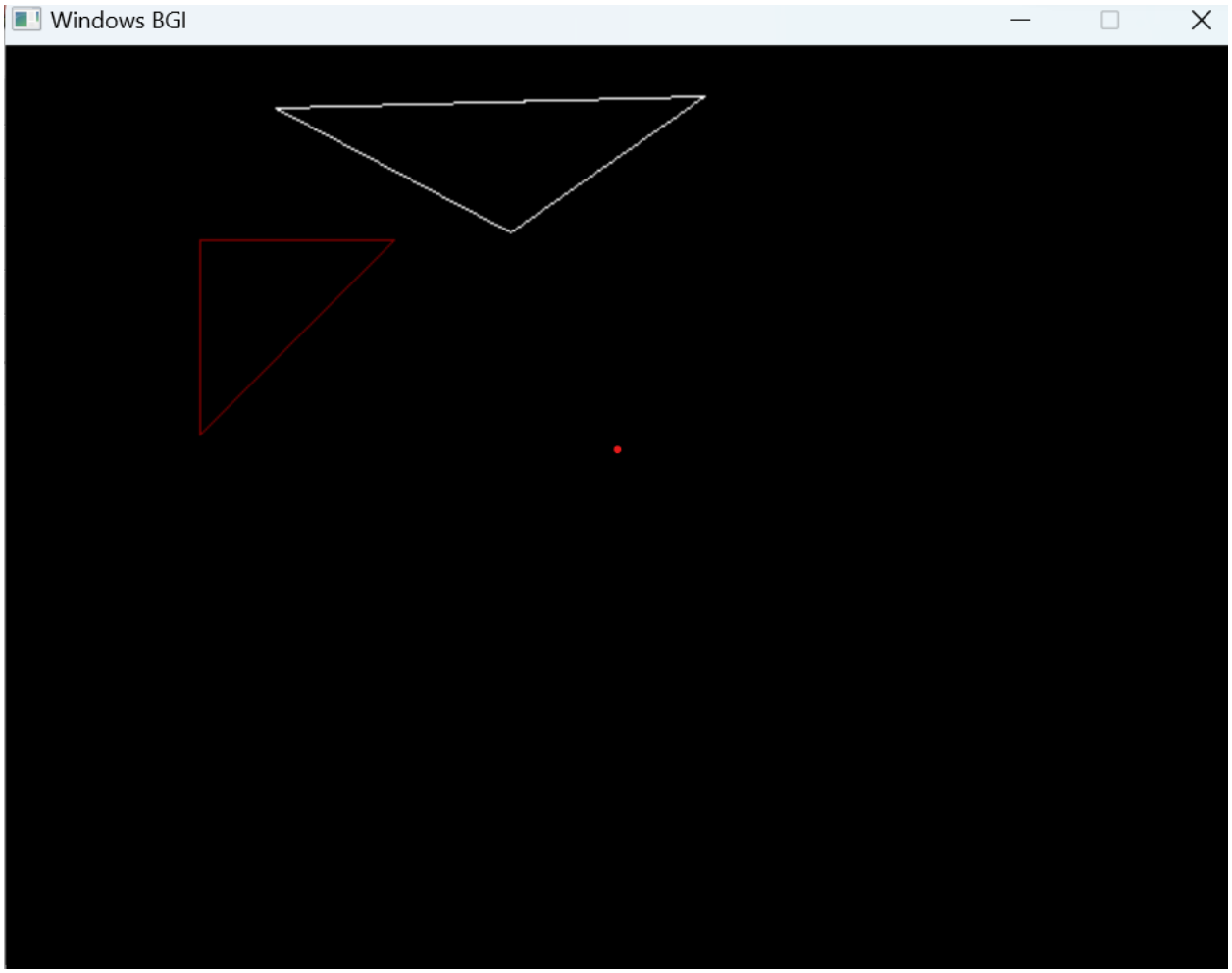
Output :




```
  else
  {
    cout << "Enter the roration angle : ";
    cin >> angle;

    cout << "Enter the pivot points : ";
```

```
    cin >> xpivot >> ypivot;

    rotation();
    setcolor(WHITE);
    drawpoly();
  }
  getch();
  closegraph();
  return 0;
}
```

Output:

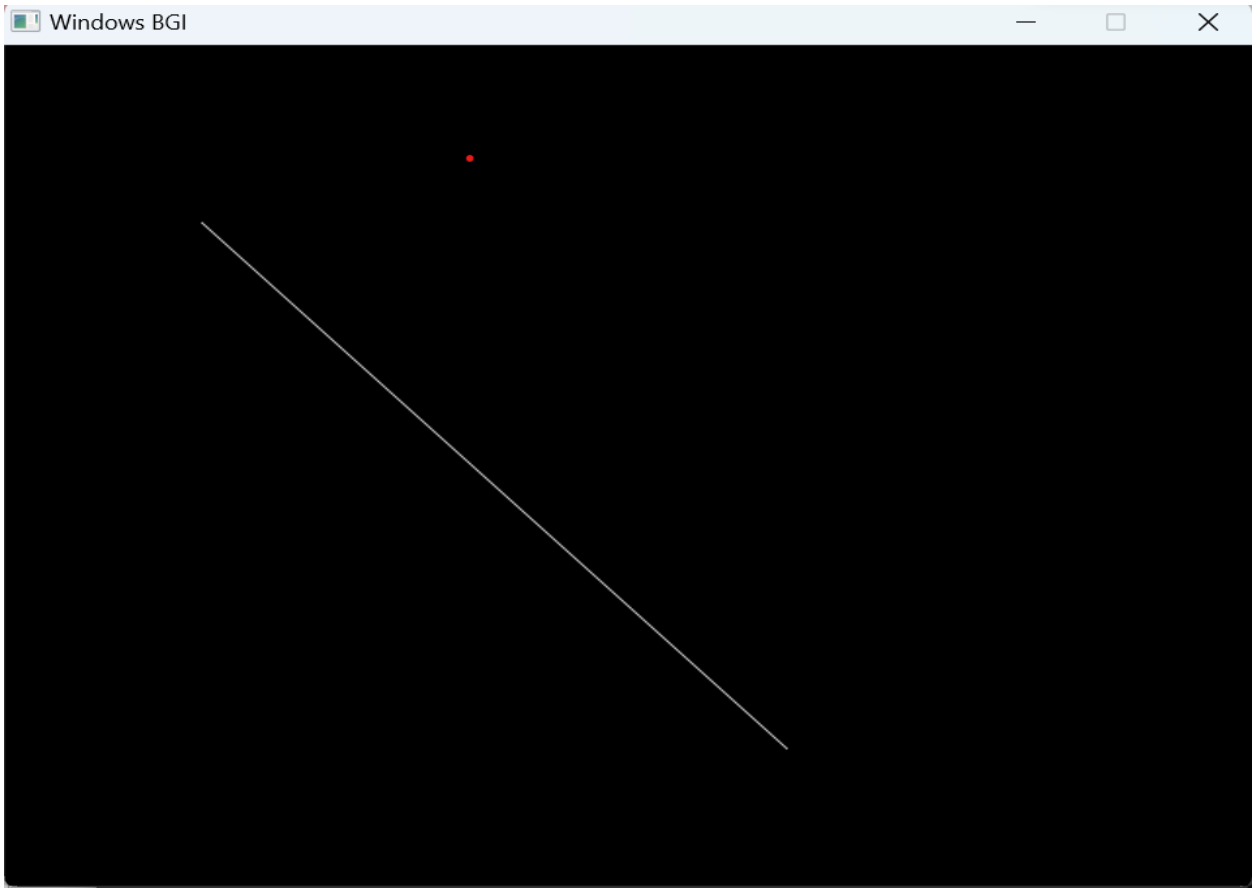


Experiment 3 : Draw a line with the Bresenham Line Drawing algorithm

```
#include<bits/stdc++.h>
#include<graphics.h>
#include<conio.h>

using namespace std;

void BresenhamLine(int x1, int y1, int x2, int y2)
{
  int dx = abs(x2-x1);
  int dy = abs(y2-y1);
  int p = 2*dy-dx;

  while(x1<x2 && y1<y2)
  {
    if(p<0)
    {
      x1++;
      p = p+2*dy;
      putpixel(x1,y1,WHITE);
    }
    else
    {
      x1++;
      y1++;
      p = p + 2*(dy-dx);
      putpixel(x1,y1,WHITE);
    }
    delay(1);
  }
}

int main()
{
```

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

    int x1, y1, x2, y2;
    cout << "Enter the coordinate of the two end point value : " << endl;
    cin >> x1 >> y1 >> x2 >> y2;

    // input : 100 100 300 300

    BresenhamLine(x1,y1,x2,y2);

    getch();
    return 0;
}
```

Output :

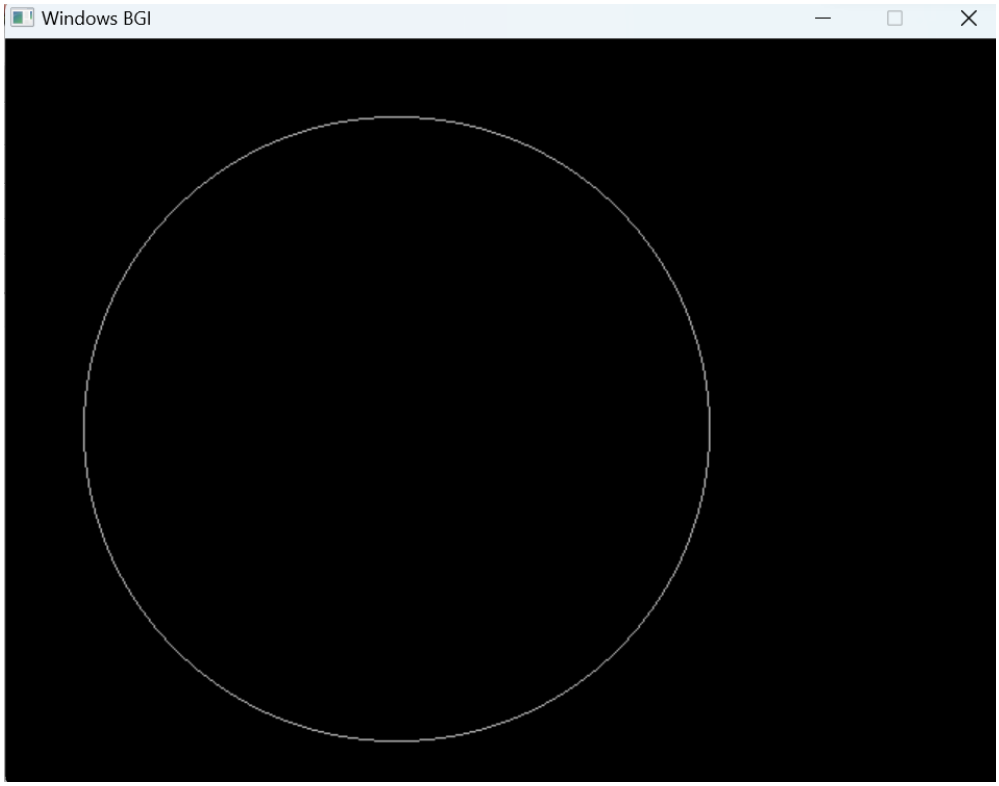


Experiment 4 : Draw a circle with the Midpoint Circle Drawing algorithm

```
#include<bits/stdc++.h>
#include<graphics.h>
#include<conio.h>

using namespace std;

void MidPointCircle(int x1, int y1, int r)
{
   int x = 0;
   int y = r;
   int p = 1-r;

   while(x<y)
   {
      putpixel(x1+x, y1+y, 7);
      putpixel(x1+x, y1-y, 7);
      putpixel(x1-x, y1+y, 7);
      putpixel(x1-x, y1-y, 7);

      putpixel(x1+y, y1+x, 7);
      putpixel(x1+y, y1-x, 7);
      putpixel(x1-y, y1+x, 7);
      putpixel(x1-y, y1-x, 7);

      x++;

      if(p<0)
      {
         p = p+2*x+1;
      }
      else
      {
```

```
        y--;
        p = p+2*x+1-2*y;
      }
      delay(1);
    }
}

int main()
{
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");

   int x, y, r;
   cout << "Enter the radius of the circle : ";
   cin >> r;    // radius = 200

   cout << "Enter the coordinate of the center(x and y) : ";
   cin >> x >> y;  // x = 250 y = 250

   MidPointCircle(x, y, r);
   getch();
   closegraph();
   return 0;
}
```

Output:



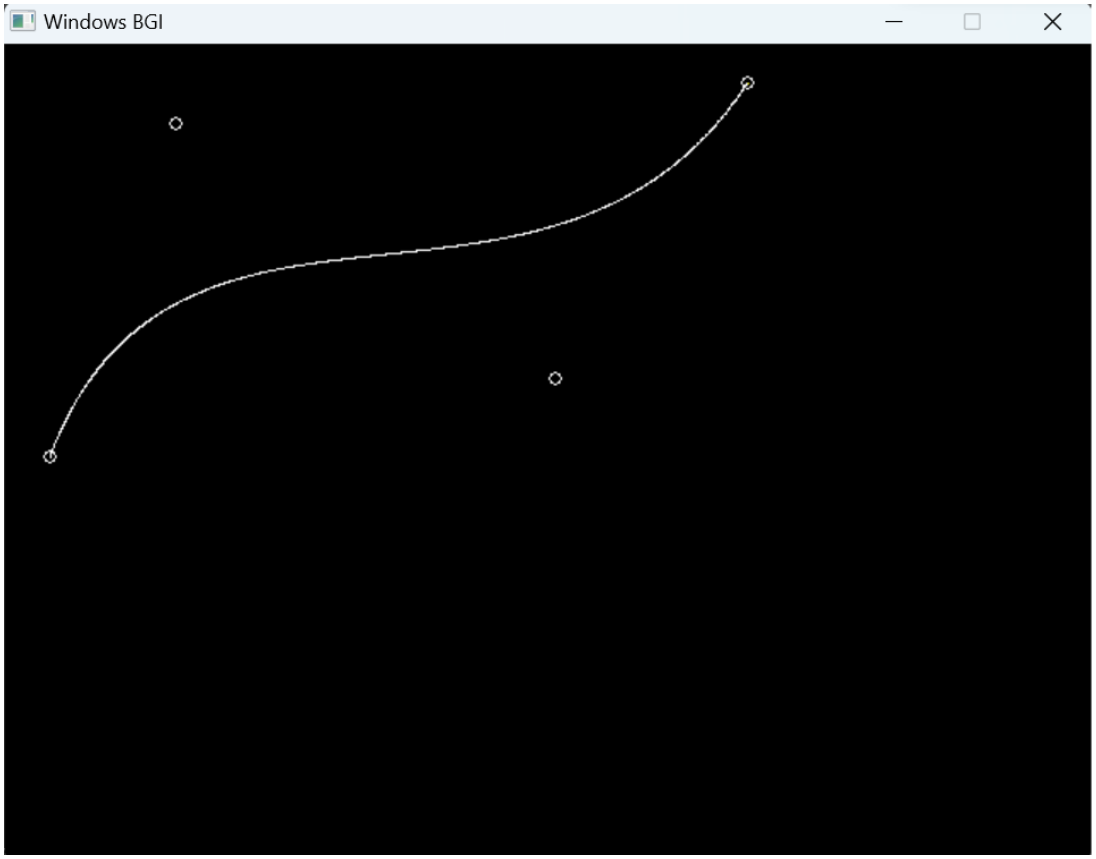
Experiment 5 : Create the Bezier Curve

```
#include<bits/stdc++.h>
#include<graphics.h>

using namespace std;

int fact(int n)
{
   if(n <= 1)
      return 1;
   return n * fact(n-1);
}

void bezier(int x[], int y[], int n)
{
   double xt, yt, u, b;
```

```
  int nfact = fact(n-1);
  putpixel(x[0], y[0], YELLOW);

  for(u=0.0; u<=1.0;u+=0.001)
  {
    xt = 0;
    yt = 0;

    for(int i=0;i<n;i++)
    {
      b = (nfact * pow(1-u, n-i-1) * pow(u,i))/(fact(n-i-1) * fact(i));
      xt += b*x[i];
      yt += b*y[i];
    }
     putpixel(xt, yt, WHITE);
  }
  putpixel(x[n-1],y[n-1], YELLOW);

  for(int i=0;i<n;i++)
  {
    circle(x[i], y[i], n);
  }
}

int main()
{
  int gd = DETECT, gm;
  initgraph(&gd, &gm, "");

  int n;
  cout << "Enter the number of contron points : ";
  cin >> n;
```

```
   int x[n], y[n];
   cout << "Enter the coordinate points : ";
   for(int i=0;i<n;i++)
   {
      cin >> x[i] >> y[i];
   }
   bezier(x, y, n);
   getch();
   closegraph();
   return 0;
}
```

Output:

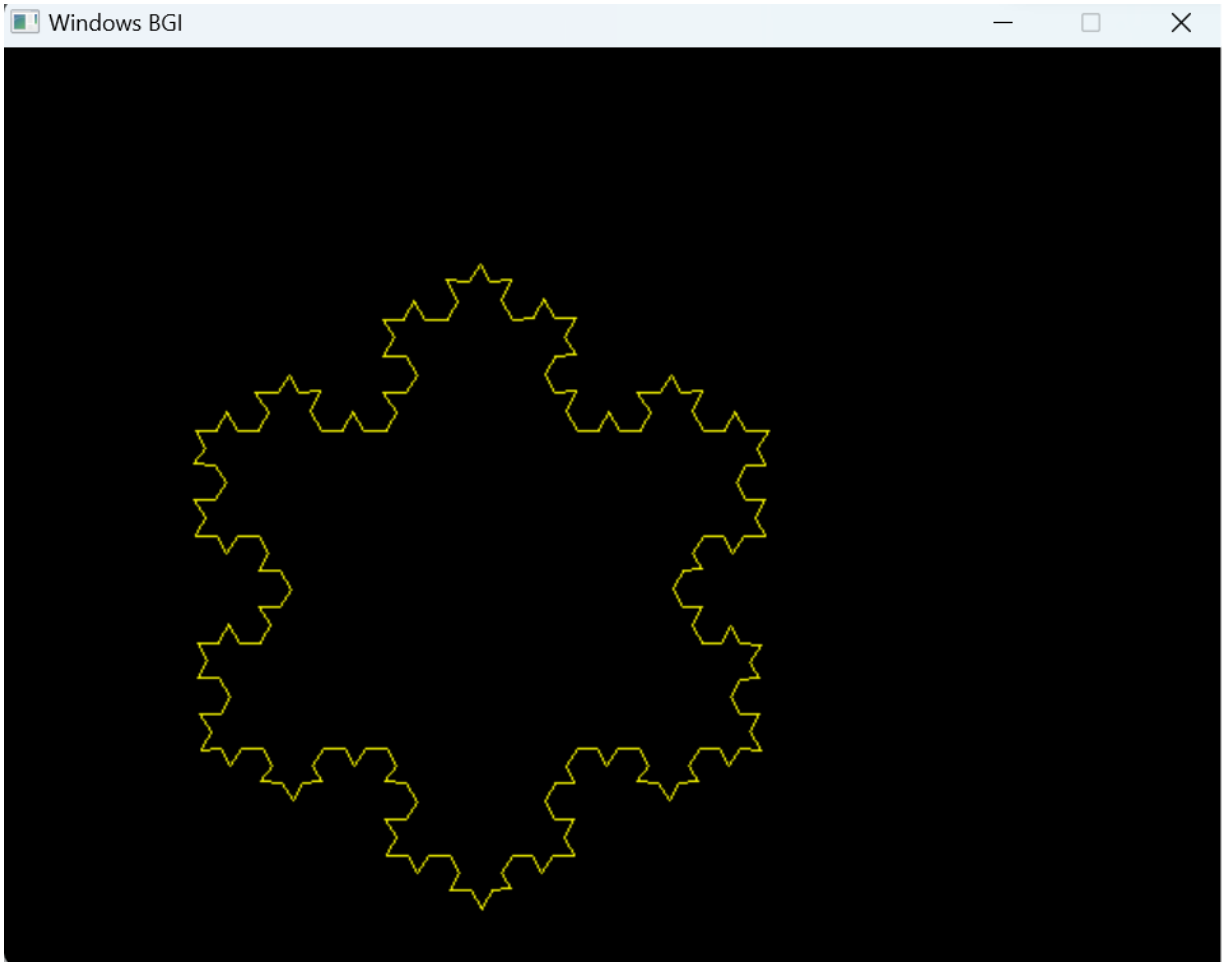


Experiment 6: Draw the Snowflake Pattern with Fractal Geometry

```
#include<bits/stdc++.h>
#include<graphics.h>
#define M_PI 3.1415962

using namespace std;

void koch_curve(int x1, int y1, int x2, int y2, int iteration)
{
  float angle = (60*M_PI)/180;

  int x3 = (2*x1 + x2)/3;
  int y3 = (2*y1 + y2)/3;

  int x4 = (x1 + 2*x2)/3;
  int y4 = (y1 + 2*y2)/3;

  int x = x3 + (x4-x3)*cos(angle) + (y4-y3)*sin(angle);
  int y = y3 - (x4-x3)*sin(angle) + (y4-y3)*cos(angle);

  if(iteration > 0)
  {
    koch_curve(x1, y1, x3, y3, iteration-1);
    koch_curve(x3, y3, x, y, iteration-1);
    koch_curve(x, y, x4, y4, iteration-1);
    koch_curve(x4, y4, x2, y2, iteration-1);
  }
  else
  {
    line(x1, y1, x3, y3);
    line(x3, y3, x, y);
    line(x, y, x4, y4);
    line(x4, y4, x2, y2);
```

```
    }
    delay(1);
}

int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    int iteration;
    cout << "Enter the number of iteration : " ;
    cin >> iteration; //iteration = 2

    int x1 = 100, y1 = 200, x2 = 400, y2 = 200, x3 = 250, y3 = 450;

    cleardevice();
    setcolor(YELLOW);

    koch_curve(x1, y1, x2, y2, iteration);
    koch_curve(x2, y2, x3, y3, iteration);
    koch_curve(x3, y3, x1, y1, iteration);

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

Output:



Experiment 7: Implement the Cohen Sutherland Line Clipping algorithm

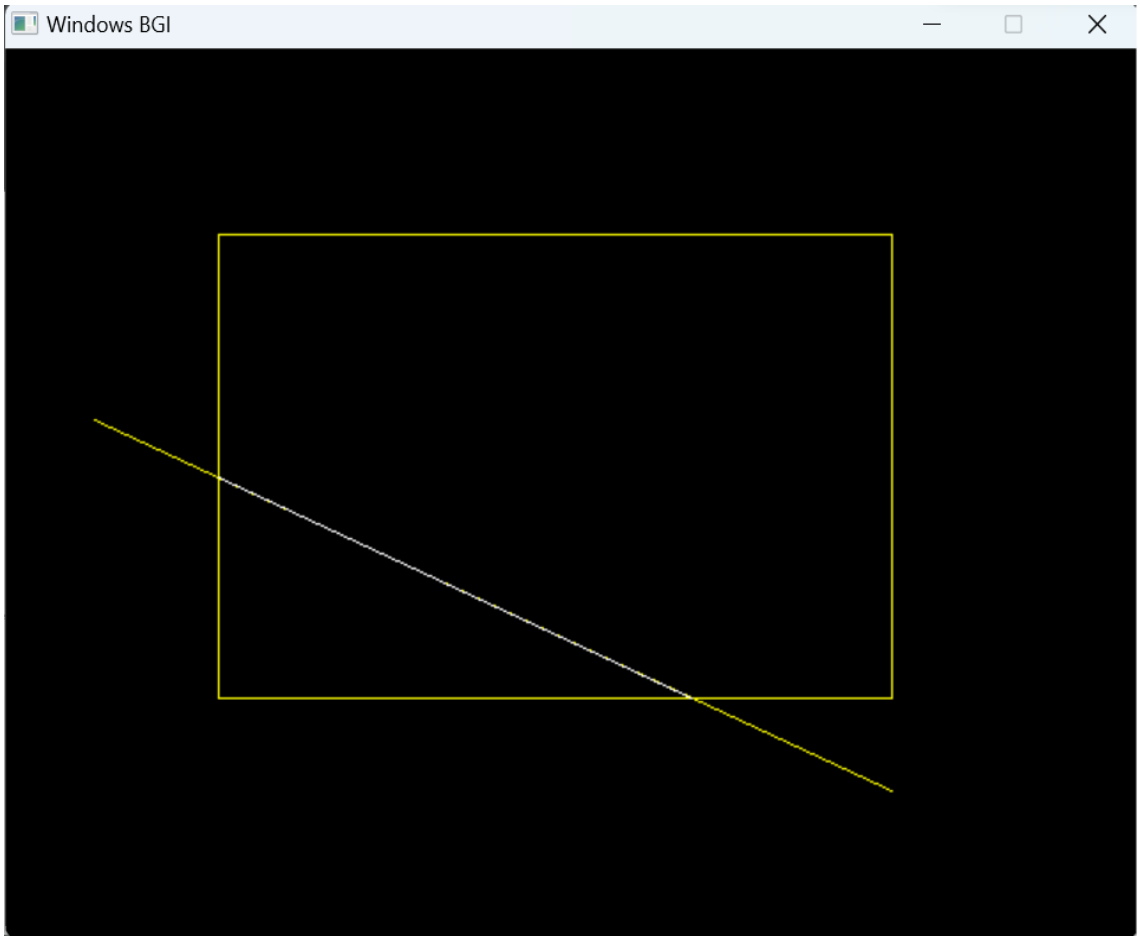
```
#include <graphics.h>
#include <bits/stdc++.h>

using namespace std;

double x_left = 120, x_right = 500, y_bottom = 100, y_top = 350;
int Left = 1, Right = 2, Bottom = 4, Top = 8;

int regionCode(int x, int y)
{
   int code = 0;
```

```
  if (x > x_right)
    code |= Right;
  else if (x < x_left)
    code |= Left;
  if (y > y_top)
    code |= Top;
  else if (y < y_bottom)
    code |= Bottom;
  return code;
}
void cohenSutherland(double x1, double y1, double x2, double y2)
{
  int code1 = regionCode(x1, y1);
  int code2 = regionCode(x2, y2);
  while (true)
  {
    double x, y;
    if (!(code1 | code2))
    {
      line(x1, y1, x2, y2);
      return;
    }
    else if (code1 & code2)
      break;
    else
    {
      int code = code1 ? code1 : code2;
      if (code & Top)
      {
        y = y_top;
        x = x1 + (x2 - x1) / (y2 - y1) * (y - y1);
      }
      else if (code & Bottom)
      {
```

```
            y = y_bottom;
            x = x1 + (x2 - x1) / (y2 - y1) * (y - y1);
        }
        else if (code & Left)
        {
            x = x_left;
            y = y1 + (y2 - y1) / (x2 - x1) * (x - x1);
        }
        else if (code & Right)
        {
            x = x_right;
            y = y1 + (y2 - y1) / (x2 - x1) * (x - x1);
        }
        if (code == code1)
        {
            x1 = x;
            y1 = y;
            code1 = regionCode(x1, y1);
        }
        else
        {
            x2 = x;
            y2 = y;
            code2 = regionCode(x2, y2);
        }
    }
  }
}
int main()
{
  int gd = DETECT, gm = DETECT;
  initgraph(&gd, &gm, "");
  setcolor(YELLOW);
  rectangle(x_left, y_bottom, x_right, y_top);
```

```
    line(50, 200, 500, 400);
    setcolor(WHITE);
    cohenSutherland(50, 200, 500, 400);
    getch();
    closegraph();
    return 0;
}
```

Output:

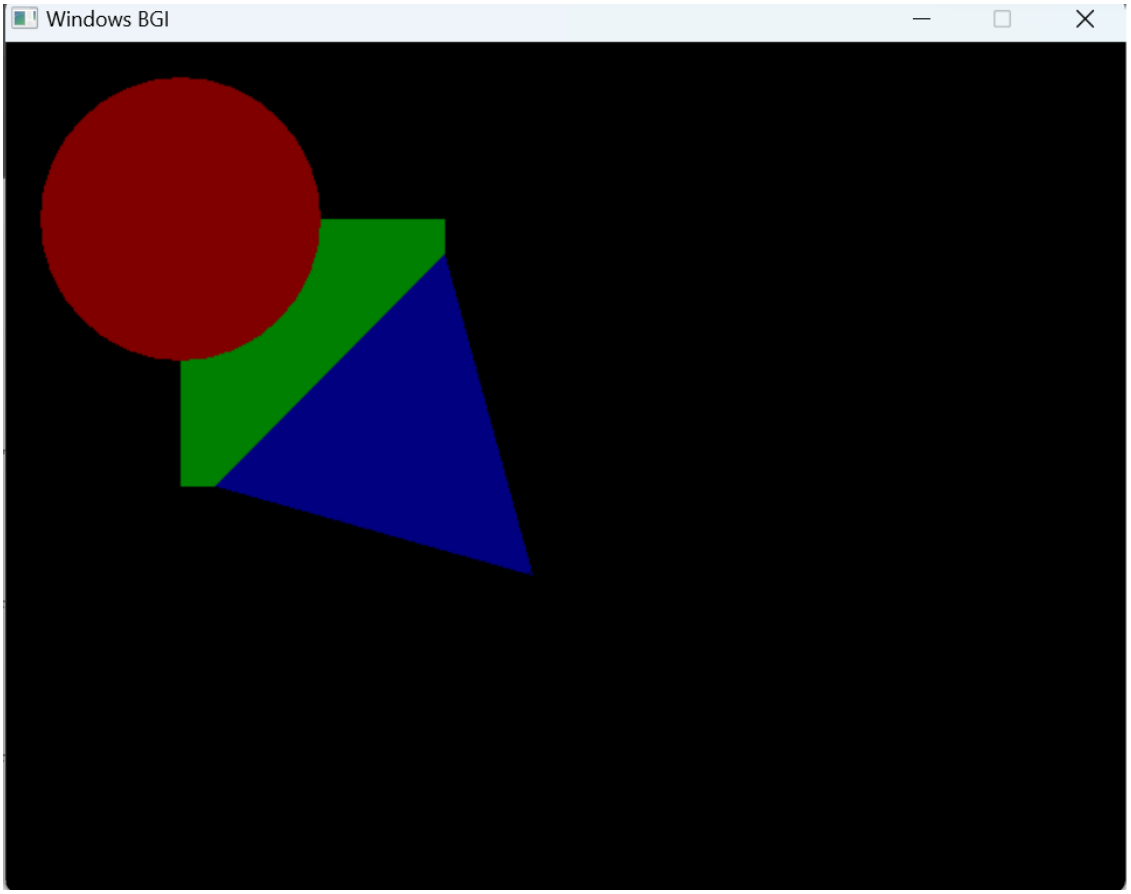


Experiment 8: Simulate Hidden Surface Elimination or Visual Surface Detection

```
#include<bits/stdc++.h>
#include<graphics.h>

using namespace std;

void circle()
{
  setcolor(RED);
  circle(100, 100, 80);
  setfillstyle(SOLID_FILL, RED);
  floodfill(100, 100, RED);
}

void rectangle()
{
  setcolor(GREEN);
  rectangle(100, 100, 250, 250);
  setfillstyle(SOLID_FILL, GREEN);
  floodfill(101,101, GREEN);
}

void triangle()
{
  setcolor(BLUE);
  line(120, 250, 250, 120);
  line(120, 250, 300, 300);
  line(250, 120, 300, 300);

  setfillstyle(SOLID_FILL, BLUE);
  floodfill(260, 260, BLUE);
}
```

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


   string sequence;
   cin >> sequence;

   for(auto x : sequence)
   {
      if(x == 'c') circle();
      else if(x == 'r') rectangle();
      else triangle();
   }

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

Output:


Windows BGI