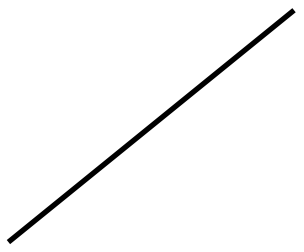
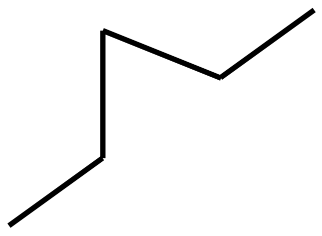
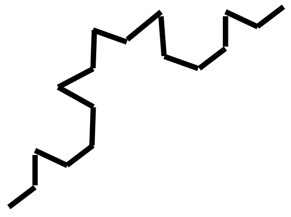
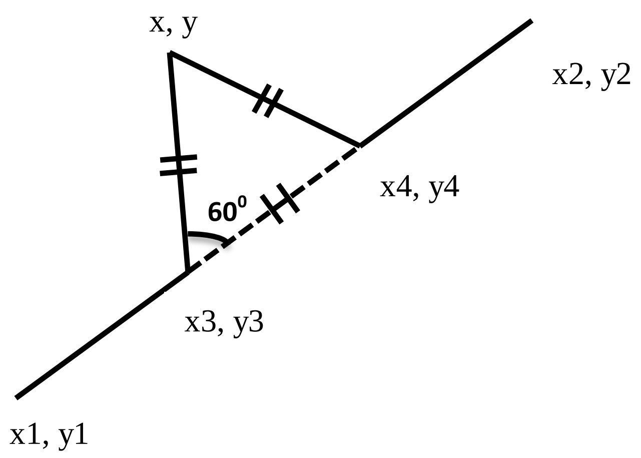
```
#include<graphics.h>
#include<stdio.h>
#include<math.h>
#include<iostream>
using namespace std;
```

/***Koch Curve:**

Iteration 0	Iteration 1	Iteration 2
		

In Iteration 0, we have a horizontal line.
In Iteration 1, line is divided into 3 equal parts. Middle part of a line is rotated in 60^0, because it forms a perfect an equilateral triangle.



Here, (x1,y1) and (x2, y2) can be either accepted from user or you can define values in program itself. You can implement any method.
Now, we can see line is divided into 3 equal segments segment((x1, y1),(x3, y3)), segment((x3, y3),(x4, y4)), segment((x4, y4),(x2, y2)) in above figure.

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

Hint: (x3, y3) is nearer to (x1, y1) than (x2, y2). So, in above formula, x1 and y1 are multiplied by 2. And we want to divide line into 3 equal segments, it is divided by 3.

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

Hint: (x4, y4) is nearer to (x2, y2) than (x1, y1). So, in above formula, x2 and y2 are multiplied by 2. And we want to divide line into 3 equal segments, it is divided by 3.

Middle segment((x3, y3),(x4, y4)) will not be drawn. But it will be rotated in 60^0 angle with respect to an arbitrary point (x3, y3). So that, we will get values of (x, y) point. Following formula will be used to computer values of (x, y).

```
float angle = 60*M_PI/180; //Convert 60⁰ angle into radians
int x = x3 + (x4-x3)*cos(angle)+(y4-y3)*sin(angle);
int y = y3 - (x4-x3)*sin(angle)+(y4-y3)*cos(angle);
```

You can accept number of iterations either from user or hardcoded value.

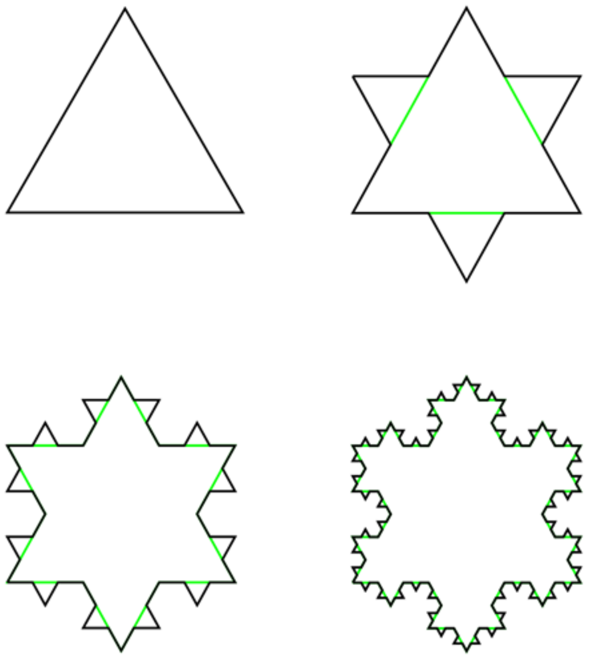
Until number iterations is not 0, we will recursively call `koch` function 4 times. Because in each iteration, each line segment is divided into 4 segments: segment((x1, y1),(x3, y3)), segment((x3, y3),(x, y)), segment((x, y),(x4, y4)) and segment((x4, y4),(x2, y2))

When number of iterations becomes 0, we will draw all segments.

```
int main(void)
{
 int gd = DETECT, gm;
 initgraph(&gd, &gm, NULL);
 int x1 = 150, y1 = 200, x2 = 290, y2 = 400;

 koch(x1, y1, x2, y2, 2);
 delay(10000);
 closegraph();
 return 0;
}

void koch(int x1, int y1, int x2, int y2, int it)
{
 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(it > 0)
 {
  koch(x1, y1, x3, y3, it-1);
  koch(x3, y3, x, y, it-1);
  koch(x, y, x4, y4, it-1);
  koch(x4, y4, x2, y2, it-1);
 }
 else
 {
  line(x1, y1, x3, y3);
  line(x3, y3, x, y);
  line(x, y, x4, y4);
  line(x4, y4, x2, y2);
 }
}
```

Snowflake Curve

```
#include<graphics.h>
#include<stdio.h>
#include<math.h>
#include<iostream>
using namespace std;
void koch(int x1, int y1, int x2, int y2, int it)
{
 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(it > 0)
 {
  koch(x1, y1, x3, y3, it-1);
  koch(x3, y3, x, y, it-1);
  koch(x, y, x4, y4, it-1);
  koch(x4, y4, x2, y2, it-1);
 }
 else
 {
  line(x1, y1, x3, y3);
  line(x3, y3, x, y);
  line(x, y, x4, y4);
  line(x4, y4, x2, y2);
 }
}

int main(void)
{
 int gd = DETECT, gm;
 initgraph(&gd, &gm, NULL);
 int x1 = 150, y1 = 200, x2 = 290, y2 = 400;
 float angle = 60*M_PI/180;
 int x3 = x1 + (x2-x1)*cos(angle)+(y2-y1)*sin(angle);
 int y3 = y1 - (x2-x1)*sin(angle)+(y2-y1)*cos(angle);

 koch(x1, y1, x2, y2, 2);
 koch(x2, y2, x3, y3, 2);
 koch(x3, y3, x1, y1, 2);
 delay(10000);
 closegraph();
 return 0;
}
```



Snowflake curve is drawn using koch curve iterations. In koch curve, we just have a single line in the starting iteration and in snowflake curve, we have an equilateral triangle. So, we just have to make changes in main() function. Draw an equilateral triangle and call koch() thrice for all three segments of an equilateral triangle.

We will consider hardcoded values of (x1, y1) and (x2, y2). Line((x1, y1) and (x2, y2)) can be rotated in 60^0 with respect to (x1, y1) point to find values of (x3, y3).

```
 int x1 = 150, y1 = 200, x2 = 290, y2 = 400;
 float angle = 60*M_PI/180;//Convert 60⁰ into radians

 int x3 = x1 + (x2-x1)*cos(angle)+(y2-y1)*sin(angle);  //rotarion
 int y3 = y1 - (x2-x1)*sin(angle)+(y2-y1)*cos(angle);  //rotarion

 //Now, we have all three coordinates of an equilateral triangle (x1,
y1), (x2, y2) and (x3, y3). Call koch() for all sides of an
equilateral triangle.
```

```
koch(x1, y1, x2, y2, 2);
koch(x2, y2, x3, y3, 2);
koch(x3, y3, x1, y1, 2);
```