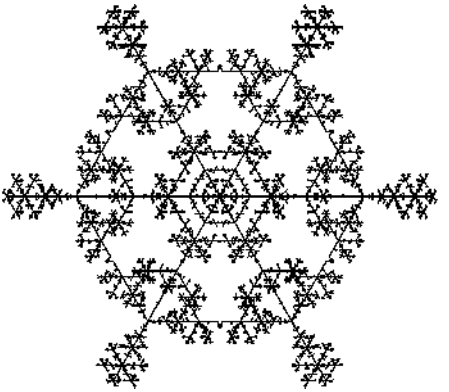
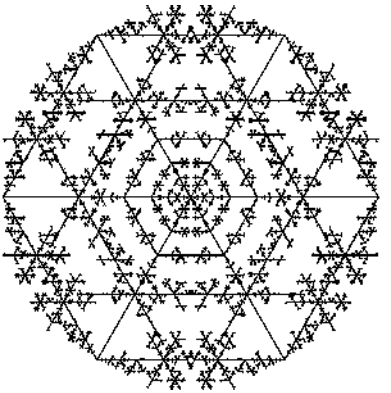


L-system Concepts



This document may be freely copied. It was initially written 54 years after Aristid Lindenmayer (1925-1989) introduced L-systems in 1968. File date for the current document: Dec 22, 2022.

The free programs *Fractint* and *Arcnel* understand the L-systems discussed in the first four chapters. The rest of the chapters introduce concepts that can be used in a special type of L-system called an *arc L-system*. These special L-systems can be understood by the free program *Arcnel*.

For additional information about L-systems, select "Help" in the "Extra" menu in *Arcnel*. To gain even more understanding, select various L-systems in the "Extra" menu and study their text. To see the text of an L-system, select "Show L-system Text" in the "Edit" menu. To see even more L-systems, select "Write Arcnel.AL" or "Write Fractint.L" in the "File" menu. After a file has been written, it can be opened by selecting "Open" in the "File" menu.

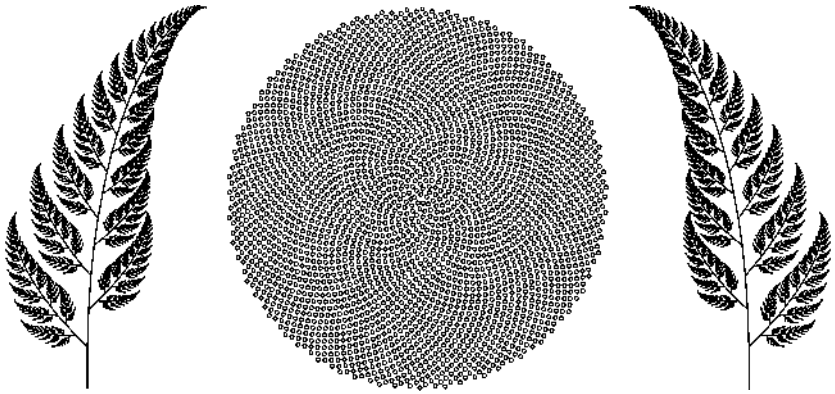
A number of the designs in this document are fractals. For example, consider the fern-like designs on the next page. A "leaf" on one of those designs looks somewhat similar to the overall design. Benoit Mandelbrot coined the word *fractal* in 1975. The Creator has designed many things that have a fractal nature.

O LORD, how great are thy works!
and thy thoughts are very deep.
Psalm 92:5

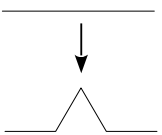
O LORD, how manifold are thy works!
in wisdom hast thou made them all:
the earth is full of thy riches.
Psalm 104:24

The works of the LORD *are* great,
sought out of all them that have pleasure therein.
Psalm 111:2

Contents



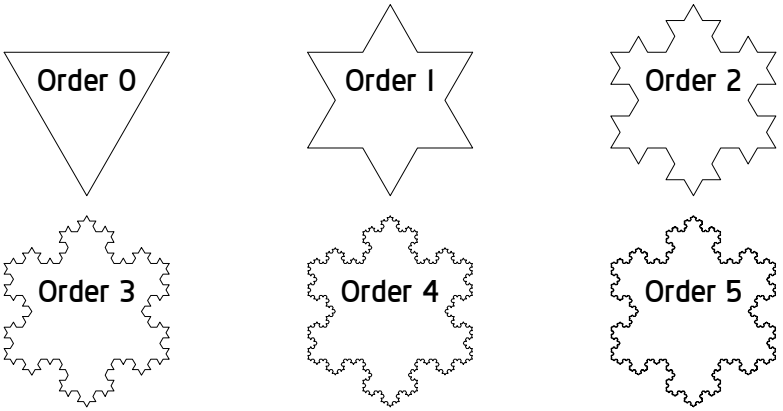
1) Basic Concepts



Consider the triangle below. It is called **Order 0**. Suppose each line segment is replaced with four smaller line segments as shown to the right.

The result is the **Order 1** diagram. Then if each smaller line segment is itself replaced with four line segments that follow the same pattern, the result is the **Order 2** diagram. Replacing each of those line segments with four smaller line segments results in the **Order 3** diagram. As the process is continued, the designs don't appear to change as much once the order is high enough.

If this process could be continued forever, the resulting design would be the Koch snowflake. Since the higher orders look a lot like each other, it is not necessary to continue forever to get some idea of what the Koch snowflake would look like.



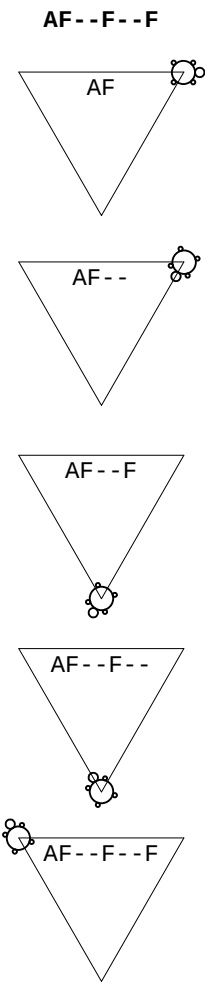
L-systems can be used to make various designs. To the right is an L-system that can be used to make the designs shown above.

```
KochFlake {
  Angle 6
  Axiom AF--F--F
  F=F+F--F+F
  A=A@I3
}
```

Imagine a turtle that crawls **f**orward when it sees '`F`'. It turns counterclockwise when it sees '`+`' and turns clockwise when it sees '`-`'. The '`Angle 6`' command in the L-system has the turtle turn $\frac{1}{6}$ of a circle when it sees '`+`' or '`-`'. It turns 60° for '`+`' and -60° for '`-`'.

To draw the **Order 0** design, the turtle reads the axiom. So the turtle sees the symbols '`AF--F--F`'. The symbol '`A`' is meaningless to the turtle, so the turtle ignores it. Then the turtle sees '`F`' so it

crawls forward 1 unit and leaves a mark as it crawls. This draws the top line of the triangle. The turtle had been at the top-left vertex of the triangle and is now at the top-right vertex as shown to the right.



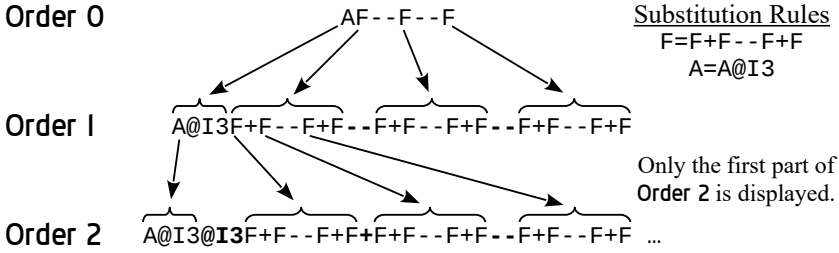
Now the turtle sees '-' and turns clockwise $\frac{1}{6}$ of a circle (that is, -60°). It sees another '-' and again turns clockwise $\frac{1}{6}$ of a circle. Its direction is -120° as shown to the right.

Then the turtle sees the 'F' so it crawls forward 1 unit and leaves a mark as it crawls. This draws the right side of the triangle. The turtle is now at the bottom vertex of the triangle as shown.

Now the turtle sees '-' and turns clockwise $\frac{1}{6}$ of a circle. It sees another '-' and again turns clockwise $\frac{1}{6}$ of a circle. Its current direction is -240° as shown.

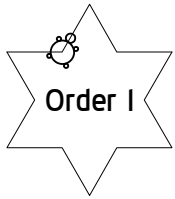
Then the turtle sees the last 'F' and crawls 1 unit and leaves a mark as it crawls. This draws the left side of the triangle. The turtle is now at the top-left vertex of the triangle as shown. This is where it started. However, the turtle is pointing a different direction than initially. Its initial direction was 0°. Its new direction is -240° which is the same direction as 120°.

To draw **Order 1**, the computer makes use of the substitution rules 'F=F+F--F+F' and 'A=A@I3'. The substitution rule 'F=F+F--F+F' says to replace each 'F' with 'F+F--F+F'. The rule 'A=A@I3' says to replace each 'A' with 'A@I3'. Study the diagram at the top of the next page to see how the axiom (the command string for **Order 0**) can be transformed into the command string for **Order 1**. Then the command string for **Order 1** can be transformed into the command string for **Order 2**, etc. For this L-system, only the symbols 'F' and 'A' get changed by substitution rules. Study the writing on the next page to see why 'A' is important even though it is meaningless to the turtle.



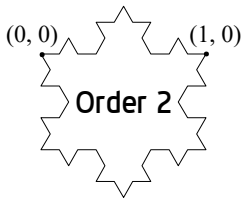
Symbols in bold came from the previous order.

The turtle ignores the '`A`' in the command string for **Order 1**. Then it sees '`@I3`'. This tells the turtle to multiply its crawl length by the **i**nverse of 3, that is, to multiply its crawl length by $\frac{1}{3}$. Since its initial crawl length is 1 unit, its new crawl length is $\frac{1}{3}$ of a unit. Then the turtle sees '`F`' and crawls forward $\frac{1}{3}$ of a unit and makes a mark as it crawls. Then it sees '`+`' and turns counterclockwise $\frac{1}{6}$ of a circle so its angle is now 60° as shown to the right. It continues reading the command string and obeys each symbol to draw the **Order 1** diagram.



When the turtle sees the command string for **Order 2**, it ignores the '`A`'. Then it sees the first '`@I3`'. So it multiplies its crawl length of 1 unit by $\frac{1}{3}$ (the inverse of 3) to make a new crawl length of $\frac{1}{3}$ of a unit. Then the turtle sees the next '`@I3`'. So it again multiplies its crawl length by $\frac{1}{3}$. Now its crawl length is $\frac{1}{9}$ of a unit. So each time the turtle sees '`F`' it crawls forward $\frac{1}{9}$ of a unit.

The diagram to the right shows the coordinates of two points on the **Order 2** design. If the command string did not have '`@I3@I3`' at the beginning, then the design would look like this **Order 2** design, but the vertex labeled as (1, 0) would be at (9, 0).



The '`KochFlake1`' L-system shown to the right results in an **Order 2** command string without '`@I3@I3`' at the start. On a computer, the design looks like '`KochFlake`' because of how the computer resizes the design to fit in a window.

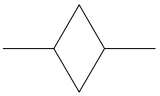
```
KochFlake1 {
  Angle 6
  Axiom F--F--F
  F=F+F--F+F
}
```

2) Turtle Memory

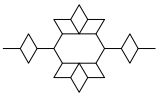
```
DoubleKoch {
  Angle 6
  Axiom F
  F=F[-F++F]+F--F+F
}
```

The '`DoubleKoch`' L-system replaces a line segment with six line segments as is displayed by the change from the **Order 0** diagram to the **Order 1** diagram shown on this page. This L-system doesn't make use of an '`@I3`' command to resize the crawl length. The computer automatically resizes the design so it fits in a window on the computer screen.

Order 0



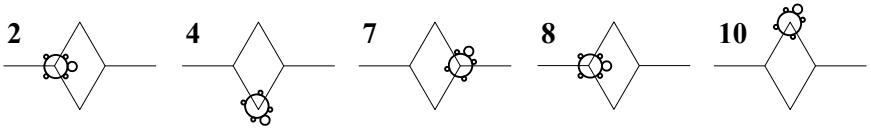
Order 1



Order 2

For **Order 0**, the only command is the '`F`' in the axiom. For **Order 1**, the rule '`F=F[-F++F]+F--F+F`' replaces that '`F`' with '`F[-F++F]+F--F+F`' which is thus the command string. After the turtle obeys the initial '`F`', it comes to '`[`'. This opening bracket tells the turtle to remember certain things. Two things it remembers are its location and direction. Then when it comes to '`]`' it returns to that location without making a mark. It also makes sure that it is pointing the remembered direction. The designs below show the location of the turtle after it has completed the displayed commands. Each design is numbered according to the number of the last command processed.

`F[` `F[-F` `F[-F++F` `F[-F++F]` `F[-F++F]+F`



When the turtle goes from command 7 to command 8, both its location and direction change. Command 8 tells it to return to the remembered condition, which is its condition at command 2 when it was told to remember its condition.

The **Order 1** command string is '`F[-F++F]+F--F+F`'. Each '`F`' in that string is replaced with '`F[-F++F]+F--F+F`' to form the **Order 2** command string. The resulting string has 99 commands. The first part is '`F[-F++F]+F--F+F`**`[`**`-F[-F++F]+F--F+F++F[-F++F]+F--F+F`**`]`**`+`' where the second '`[-F++F]`' sequence is underlined. This sequence is between bold brackets.

A condensed form of the string is '…F**[**-F[-F++F]+…**]**+…'. The turtle remembers its condition when it comes to the opening bold **bracket**. Then at the opening underlined bracket, it remembers its new condition. Then when it comes to the closing underlined bracket, it returns to the condition it had at the opening underlined bracket. Then when it comes to the closing bold **bracket**, it returns to the condition it had at the opening bold **bracket**.

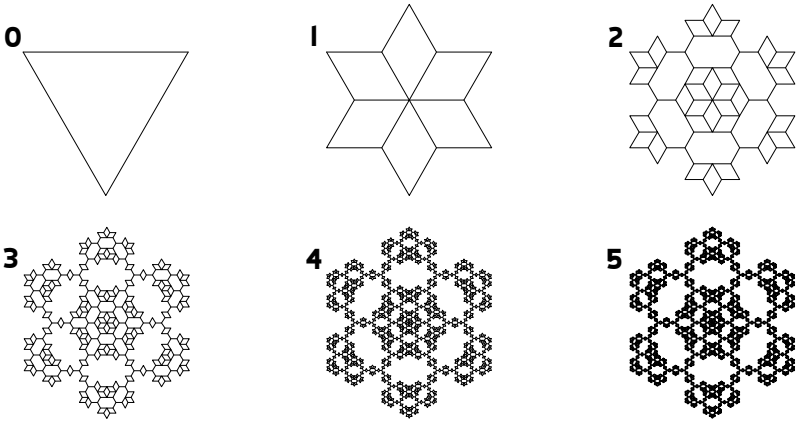
When the turtle comes to the opening underlined bracket, it stacks information on top of information that has already been stored. The turtle *pushes* information onto the top of the stack. When the turtle comes to the closing underlined bracket, it removes that information from the top of the stack. It *pops* information off the stack. An opening bracket is a *push* command, and a closing bracket is a *pop* command.

The rule 'F=F[-F++F]+F--F+F' is used in 'KochFlake2' to the right to make a variation of the Koch snowflake. Compare that rule to 'F=F+F--F+F' which was used in 'KochFlake1'.

```
KochFlake2 {
  Angle 6
  Axiom F--F--F
  F=F[-F++F]+F--F+F
}
```

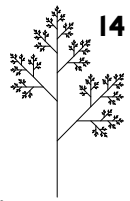
For **Order 0**, the commands in the axiom are used. Then for **Order 1**, each 'F' in the axiom is replaced with 'F[-F++F]+F--F+F' to produce 'F[-F++F]+F--F+F--F[-F++F]+F--F+F--F[-F++F]+F--F+F'. Then each 'F' in that string is replaced with 'F[-F++F]+F--F+F' to make the command string for **Order 2**, etc. The numbers with the designs below are the numbers of the orders. The list below shows the number of commands in the command string for each order.

0) 7 **1**) 49 **2**) 301 **3**) 1813 **4**) 10,885 **5**) 65,317

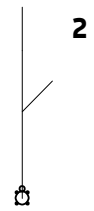


3) Reverses

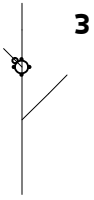
```
Plant45 {
  Angle 8
  Axiom ++FA
  F=@Q2F@IQ2
  A=[S]F!A
  S=-FA
}
```



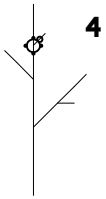
Order 14 of '`Plant45`' produces the design shown to the far right. '`Angle 8`' has the turtle turn $\frac{1}{8}$ of a circle when it sees '+' or '-'. So it turns 45° at a '+'. The command string for **Order 2** begins with '`++@Q2@Q2F@IQ2@IQ2[-FA]@Q2F@IQ2![S]F!A`'. Each '+' tells the turtle to turn counterclockwise 45°. So '++' positions the turtle at 90° as shown.



The command '`@Q2`' tells the turtle to multiply its crawl length by the s**q**uare root of 2. The command '`@IQ2`' tells the turtle to multiply its crawl length by the **i**nverse of the s**q**uare root of 2. This is equivalent to dividing its crawl length by the square root of 2. Once it obeys the commands '`++@Q2@Q2F@IQ2@IQ2`', its crawl length is again 1 unit since it has twice multiplied by $\sqrt{2}$ and twice multiplied by the inverse of $\sqrt{2}$.



One part of the **Order 3** command string is '`![-FA]`'. The '`!`' tells the turtle to reverse the meanings of '+' and '-'. Normally '-' means to turn *clockwise*. Now that the meanings have been reversed, the '-' after the '`!`' means to turn *counterclockwise*. Once the turtle turns, it will be positioned as shown, ready to draw the stem at that point.

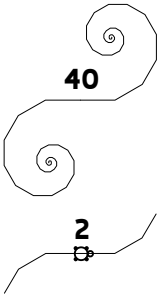


The string '…`!`…`![-FA]`…' is a condensed form of the **Order 4** command string. Each '…' indicates omitted symbols. The first '`!`' tells the turtle to reverse the meanings of '+' and '-'. The second '`!`' tells the turtle to again reverse the meanings of '+' and '-'. So they are now restored to their original meanings. Thus '-' has its normal meaning of turning *clockwise*. Once the turtle turns, it will be positioned as shown, ready to draw the stem at that point.

The command '`[`' tells the turtle to remember whether the meanings of '+' and '-' are normal or reversed. Then at the corresponding '`]`', the turtle returns to the understanding of '+' and '-' it had at '`[`'. This way, any '`!`' between brackets will not affect the turtle's understanding after the brackets.

Order 40 of '`DoubleSpiral`' is shown to the far right. The '`Angle 12`' command tells the turtle to turn $\frac{1}{12}$ of a circle when it sees '`+`' or '`-`'. So it turns 30° counterclockwise when it sees '`+`'. The turtle multiplies its crawl length by 0.9 if it sees '`@.9`'. The command string for **Order 2** is '`[F+@.9F+@.9FA]|F+@.9F+@.9FA`'. The initial position of the turtle is as shown.

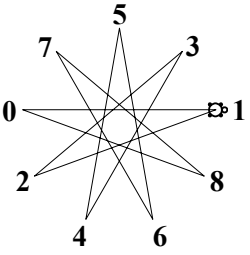
```
DoubleSpiral {
  Angle 12
  Axiom [FA]|FA
  A=+@.9FA
}
```



The '`[`' tells the turtle to remember that position as well as the initial crawl length. When the turtle comes to the '`]`' it returns to that initial position and crawl length. Then '`|`' (vertical line) tells the turtle to reverse its direction. So it now points to the left instead of to the right. It is now ready to draw the left side of the design. On a keyboard, '`|`' may look like '`¦`'.

To the far right is **Order 8** of '`Star9`' with the command string underneath. The number at each vertex is the number of '`F`' commands the turtle needs to obey to get to that vertex.

```
Star9 {
  Angle 9
  Axiom FA
  A=|FA
}
```



Order 8

F|F|F|F|F|F|F|F|FA

The turtle starts at vertex 0. The first '`F`' tells the turtle to crawl to vertex 1 as shown. Then '|' tells the turtle to reverse direction. However, '`Angle 9`' tells the turtle to turn in increments of $\frac{1}{9}$ of a circle. So the allowed directions for the turtle are 0°, 40°, 80°, 120°, 160°, 200°, 240°, 280°, and 320°. The turtle is currently pointing in the 0° direction. When the turtle turns 180°, it is not pointing an allowed direction. So the turtle keeps turning counterclockwise until it comes to the 200° direction.

The next '`F`' tells the turtle to crawl to vertex 2. The next '|' tells the turtle to reverse direction. Since the turtle is pointing in the 200° direction, adding 180° would make 380° which is the same direction as 20°. However, 20° is not allowed. So the turtle turns counterclockwise until it comes to 40°. Thus it turns counterclockwise a total of 200°. When '`Angle`' has an odd number, then '|' tells the turtle to turn 180° counterclockwise and keep turning until it comes to the next allowed angle.

4) Additional Angles

To the far right is **Order 28** of '`PlantTilt`' which uses '`D`' instead of '`F`' to tell the turtle to **d**raw line segments. To tell the turtle to turn for the '`D`' command, '`\`' and '`/`' are used instead of '`+`' and '`-`'. Normally, '`\90`' tells the turtle to turn 90° to the *left*, and '`/45`' tells the turtle to turn 45° to the *right* (unless '`!`' reverses meanings).

```
PlantTilt {
  Angle 4
  Axiom \90DA
  D=@1.2D@I1.2
  A=[@1.2S][!S]\3DA
  S=T
  T=U
  U=V
  V=W
  W=X
  X=/45DA
}
```

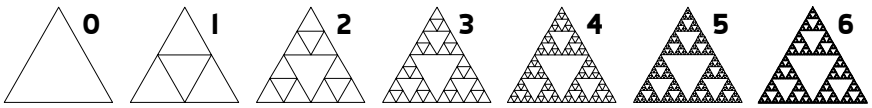
Order 7 of this L-system has '`[!/45DA]`' in the command string. Because of the '`!`', the turtle turns 45° to the *left* when it sees '`/45`'. Then '`D`' tells it to draw the little stem on the left of the main stem (see the **Order 7** diagram above). Then at the '`]`', the meanings of '`\`' and '`/`' are restored to what they were at the '`[`'.

Sometimes it is useful to tell the turtle to go forward without leaving a mark. The commands '`G`' and '`M`' are for that purpose. The commands '`+`' and '`-`' change the angle for '`G`' which tells the turtle to **g**o forward without leaving a mark. The commands '`\`' and '`/`' change the angle for '`M`' which tells the turtle to **m**ove forward without leaving a mark.

The turtle keeps track of two angles. One is for '`F`' and '`G`'. The other is for '`D`' and '`M`'. The commands '`+`', '`-`', and '`|`' change the angle for '`F`' and '`G`'. The commands '`\`' and '`/`' change the angle for '`D`' and '`M`'. To the right are L-systems that use '`FG`' and '`DM`' commands. For the '`DM`' L-system, '`Angle 3`' could be '`Angle 4`' since the '`Angle`' command doesn't affect '`D`' and '`M`'. The designs below can be made with both the '`FG`' and the '`DM`' L-systems.

```
SierpinskiFG {
  Angle 3
  Axiom F+F+F
  F=F[+F+F]G
  G=GG
}

SierpinskiDM {
  Angle 3
  Axiom D\120D\120D
  D=D[\120D\120D]M
  M=MM
}
```

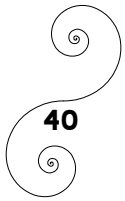


Arcs

```
DoubleSpiralA {~
  Angle 12
  Axiom [LA]|LA
  A=@.9LA
}

DoubleSpiralA2 {~
  Axiom [(30A]\180(30A
  A=@.9(30A
}
```

The program *Arcnel* was designed to understand the same L-systems that the free program *Fractint* understands. *Arcnel* also understands an additional type of L-system called an *arc L-system*. Arc L-systems can use symbols for drawing arcs of circles. As shown to the right, a '~' (tilde) immediately after the opening brace says that an L-system is an arc L-system.



The **Order 40** design to the right can be made using either arc L-system shown above. '`DoubleSpiralA`' uses '`FG`' commands. '`DoubleSpiralA2`' uses '`DM`' commands. For both types of commands, the arc is a portion of a circle whose radius is the current crawl length.

The '`FG`' commands '`L`' and '`R`' tell the turtle to draw an arc as it turns to the **l**eft (counterclockwise) or to the **r**ight (clockwise), unless '`!`' has reversed the meanings of '`L`' and '`R`'. The fraction of the circle drawn is set by the '`Angle`' command. So '`Angle 12`' tells the turtle to draw an arc that is $\frac{1}{12}$ of a circle.

The '`DM`' commands for drawing arcs are '`(`' and '`)`'. Normally, the '`(`' and '`)`' commands tell the turtle to draw an arc while turning to the left (counterclockwise) or to the right (clockwise). So '`(30`' tells the turtle to draw a 30° arc while turning to the left, unless '`!`' has reversed the meanings of '`(`' and '`)`'.

Not only do arc L-systems have arc-drawing ability, they have other additional features. For example, if no '`F`', '`G`', '`L`', '`R`', '`+`', '`-`', or '`|`' commands are used, then no '`Angle`' command is needed. The following pages discuss additional features of arc L-systems. These additional abilities can make it useful for an L-system to be an arc L-system even if no arcs are drawn.

At times, a standard L-system might make use of the symbols '`L`' and '`R`'. For such an L-system, the turtle ignores those symbols. If a '~' (tilde) were placed after the opening brace, the turtle would no longer ignore those symbols. So if a standard L-system is changed into an arc L-system, those symbols should be changed to something else, such as '`U`' and '`V`'.

6) Delayed Substitution

```
Leaf130cS {
  Angle 8
  Axiom ++A
  F=@1.13F@I1.13
  A=F[+S]F!A
  S=T
  T=U
  U=V
  V=W
  W=X
  X=Y
  Y=A
}

Leaf130c {~
  Angle 8
  Axiom ++A
  F=@1.13F@I1.13
  A=F[+:7&A]F!A
}
```

Each L-system to the right can make the **Order 46** design shown below. This design is like one on p. 130 of *The Algorithmic Beauty of Plants* by Przemyslaw Prusinkiewicz and Aristid Lindenmayer. The second man listed is the one for whom L-systems (Lindenmayer systems) are named. L-systems help us to somewhat mimic beautiful designs the Creator has placed in plants.

These L-systems make use of delayed substitution. The arc L-system '`Leaf130c`' uses seven fewer substitution rules than the standard L-system '`Leaf130cS`'. It can do this because the symbols '`:`' and '`&`' have special meanings in arc L-systems.

The symbols '`:7&A`' are in the **Order 1** command string. The turtle ignores those symbols. Then when the computer applies substitution rules, it changes '`:7`' to '`:6`'. The '`&`' tells it to ignore the next symbol which is '`A`'. So the computer does not apply the '`A`' substitution rule. Thus the **Order 2** command string has '`:6&A`'.

The **Order 3** command string has '`:5&A`' since the ':' told the computer to decrease the value of '`6`' by one. Also, the '`&`' told the computer to ignore the '`A`'.

The **Order 4** command string has '`:4&A`'. The **Order 5** command string has '`:3&A`'. The next command strings have '`:2&A`' and '`:1&A`' and '`:&A`' (which is equivalent to '`:0&A`'). When the computer processes '`:&A`' as it makes the next command string, the ':' tells it to ignore and delete the next symbol which is '`&`'. Also, the ':' is deleted. Since the '`&`' is now ignored, it no longer tells the computer to ignore the '`A`' . So the '`A`' substitution rule is applied.

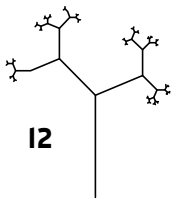
The discussion above shows that '`:7&`' in '`Leaf130c`' results in seven delays of the implementation of the '`A`' substitution rule. So the last eight substitution rules in '`Leaf130cS`' can be reduced to one substitution rule by using special symbols available in arc L-systems.

Choices

```
PlantC {~29851733
  Angle 16
  Axiom ++++FK
  K=,9,1
  K=[BT]!BT,AT
  B=,1,1
  B=++,+++
  A=,1,1,1
  A=+,-,
  T=,1,1,1
  T=@.4FK,@.5FK,@.6FK
}
```

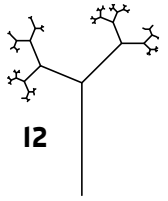
In ‘PlantC’, the ‘K’ substitution rule is spread over two lines. It could be written as ‘K=,9,1[BT]!BT,AT’. In an arc L-system, a ‘,’ (comma) after ‘=’ means that the substitution rule has options. The first *two* commas have numbers after them. This indicates that the substitution rule has *two* options. The first option is ‘[BT]!BT’. The second option is ‘AT’.

The numbers ‘9’ and ‘1’ tell the computer how to choose between the two options. When the computer replaces a ‘K’, the first option is 9 times more likely to be chosen than the second option.



The ‘B’ substitution rule also has two options. It could be written ‘B=,1,1++,+++’. The options ‘++’ and ‘+++’ are equally likely to be chosen.

The ‘A’ substitution rule has three options. It could be written ‘A=,1,1,1+,-,’. The three options are equally likely to be chosen. The first two options are ‘+’ and ‘-’. The third option is to substitute nothing for ‘A’. In other words, the third option is to delete ‘A’.



76344122

The ‘T’ substitution rule also has three options. The options ‘@.4FK’, ‘@.5FK’, and ‘@.6FK’ are equally likely to be chosen.

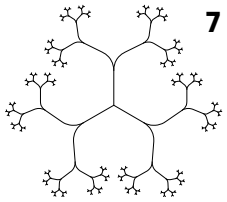
An L-system that has one or more substitution rules with options is a *stochastic* L-system. The program *Arcnel* chooses between options by using a mathematical algorithm that generates pseudorandom numbers. The ‘PlantC’ L-system has the number ‘29851733’ immediately after the ‘~’. That number is the *seed* that initializes the pseudorandom number generator. Use of a different generator would likely result in a different design for the same seed. If no number is after the ‘~’, then the computer “randomly” chooses the seed.

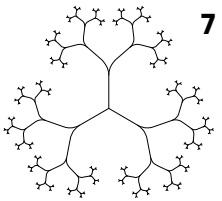
The design for **Order 12** of the ‘PlantC’ L-system is shown immediately under the L-system written in the box above. The second design shown is produced when the seed is 76,344,122.

Randomness

```
TriSym {~
  Axiom ?5641195\90[DK]\120[DK]\120DK
  K=@?.8@.6[TK]!TK
  T=(35(?30D
}
```

Not only can the computer make choices in certain substitution rules, the turtle can also make choices in obeying certain commands. In the axiom above is the command '`?5641195`'. This gives the turtle a seed for the pseudorandom number generator. Thus the "randomly" chosen seed is replaced with 5,641,195. The top design to the right is **Order 7** of the '`TriSym`' L-system. The other design comes from changing '`5641195`' in the axiom to '`94434272`'.





94434272

The command '`@?.8`' tells the turtle to choose a number from 1 down to (but not including) 0.8 and to multiply its crawl length by that number. If the turtle were given the command '`@?1.5`', it would choose a number from 1 up to (but not including) 1.5 and would multiply its crawl length by that number. If it were given the command '`@??1.5`', it would choose a number from 1 up to 1.5 and choose whether to multiply by that number or its inverse.

The command '`(?30`' tells the turtle to choose an arc measure from 0° up to (but not including) 30°. The turtle than draws an arc of that measure while turning to the left (counterclockwise), unless '`!`' has reversed the meanings of '`(`' and '`)`'. If the turtle were given the command '`(??30`', then it would choose an arc measure smaller than 30° and choose whether to turn to the left or the right.

The turtle uses the pseudorandom number generator to help it make choices. Each time the turtle makes a choice, the *state* of the pseudorandom number generator changes. When the turtle comes to a '`[`', it remembers the state of the pseudorandom number generator. At the corresponding '`]`', the state of the pseudorandom number generator is restored. So if the turtle were given the commands '`[@?2D]\30@?2D`', it would make the same choice at each '`@?2`' and would crawl the same distance at each '`D`'.

If the turtle is given the commands '`[@?2D?.]\30@?2D`', then '`?.`' tells the turtle to store the current state of the generator at the most recent stack location. So the memory the turtle stored on the stack at '`[`' is replaced with a new memory. This new memory will be the state of the generator after '`]`'. So the state of the generator at the second '`@?2`' will be different from its state at the first '`@?2`'. Thus the turtle will likely make different choices at each '`@?2`'.

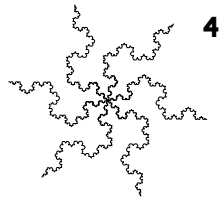
If the turtle is told '`[@?2D]?\30@?2D`', then the '`?`' after the '`]`' tells the turtle to ask the generator for a number. The turtle ignores the number. However, the state of the generator changes when it generates the number. So it is in a different state at each '`@?2`'.

If the turtle is told '`[@?2D[@?2D?..]]\30@?2D`', then '`?..`' tells the turtle to store the current state of the generator at the next to most recent stack location. So when the turtle comes to the second '`]`', the state of the generator that is remembered is the state it was in at the '`?..`' inside the innermost pair of brackets.

For each arc L-system there are two seeds. One seed prepares the computer to make choices in substitution rules. That seed can be specified by placing it immediately after the '`~`' after the opening brace. The other seed prepares the turtle for the choices it may need to make at certain commands. That seed can be replaced at the beginning of the axiom by using '`?`' followed a number.

In '`KochR`', the seed for the turtle is set to 3 by '`?3`'. When the turtle sees '`@??.8`', it chooses a number from 1 down to 0.8 and chooses to multiply by that number or its inverse. So the multiplier is between 0.8 and 1.25 (the inverse of 0.8). The command '`@I@`' tells the turtle to multiply the crawl length by the inverse of the previous multiplier. This returns the crawl length to what it had been before '`@??.8`'.

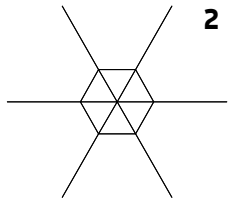
```
KochR {~
  Angle 6
  Axiom ?3[F]+[F]+[F]+[F]+[F]+F
  F=F+@??.8F--F@I@+F
}
```

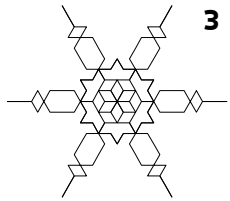


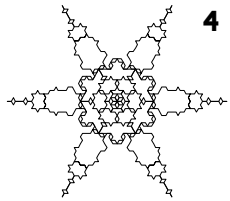
In an '`@`' expression in an arc L-system, the symbol '`@`' can be used in place of a number to represent the previous multiplier. So '`@@`' tells the turtle to multiply its crawl length by the previous multiplier, and '`@Q@`' tells the turtle to multiply its crawl length by the square root of the previous multiplier.

9) Special Substitutions

```
FlakeK {~46704894
  Angle 6
  Axiom %$F$%[A][!A]+[A][!A]+[A][!A]+[A][!A]+[A][!A]+[A][!A]+[A]!A
  A=:2:1&~*1&~A
  F=,5,1,1,1,1F+F--F+F,+F-F-F+F,+F--F+FF,+F-FF-F+,FFF
}
```







'FlakeK' is a stochastic L-system. The 'F' substitution rule has 5 options that are chosen in a probability ratio of 5:1:1:1:1. For **Order 1** there is nothing to draw. The command string begins with '%$+F--F+FF$%[:2:1&~*1&~A]'. This shows that the 'F' in the axiom was replaced with the third option, '+F--F+FF'.

The turtle ignores all symbols from the first '%' to the second '%'. This is the only portion of the string that has any draw commands. That is why there is nothing for the turtle to draw.

When the computer applies substitution rules to the **Order 1** string to make the **Order 2** command string, it sees the '*1' and does a special substitution. It searches the **Order 1** string for the first pair of dollar signs. It replaces '*1' with the symbols between that first pair. So '+F--F+FF' is substituted for '*1'.

The sequence ':2:1&~*1&~A' occurs 12 times in the **Order 1** string. So ':1:&~+F--F+FF&~:2:1&~*1&~A' occurs 12 times in the **Order 2** string. Each ':2' is changed to ':1', and each ':1' is changed to ':' which is equivalent to ':0'.

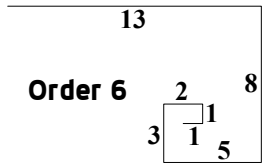
When the computer analyzes ':1:&~+F--F+FF&~' as it makes **Order 3**, it changes ':1' to ':'. It also ignores and deletes the first '&' because of the ':' which is also deleted. Since the first '&' is now ignored, it no longer tells the computer to ignore the '~' after it. A '~' (tilde) tells the computer to ignore and delete all the characters after it up through the next '~'.

So the sequence ':1:&~+F--F+FF&~' in the **Order 2** string is changed to ':~' in the **Order 3** string. Then when the **Order 4** string is made, ':~' is deleted since ':' is deleted along with '~' after it.

While these deletions are occurring, the '`F`' substitution rule keeps being applied in the section of the command string that is between the dollar signs. Also, the '`A`' substitution rule keeps being applied 12 times after the dollar signs. So each command string starting with **Order I** has '`:2:1&~*1&~A`' in it 12 times.

Each '`*1`' in a command string causes the symbols between the first two dollar signs to be substituted for the '`*1`' as the next command string is made. If there is not a pair of dollar signs, or if there is nothing between the first pair of dollar signs, then '`*1`' is left unchanged. If '`*2`' and a second pair of dollar signs are in a command string, then '`*2`' is replaced by the symbols between that second pair of dollar signs. That is, the symbols between the third '`$`' and the fourth '`$`' are substituted for '`*2`'.

```
FibonacciR {~
  Angle 4
  Axiom $F$+$FN
  S=$
  N=$+S**1**2N
}
```



In a command string, '`**1`' is replaced by the symbols between the first pair of dollar signs *before* '`**1`'. The sequence '`**2`' is replaced by the symbols between the second pair of dollar signs *before* '`**2`'. For the '`FibonacciR`' L-system, the **Order 4** command string is '`$F$+$F$+$FF$+$FFF$+$FFFFF$+S**1**2N`'. When the next order is made, '`FFFFF`' replaces '`**1`', and '`FFF`' replaces '`**2`'.

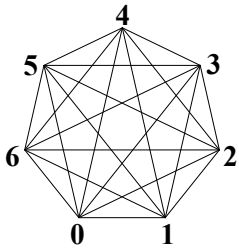
If '`*`' (asterisk/star) is not followed by a number, then the symbols between the first pair of dollar signs are placed after the '`*`' which is left unchanged. If '`**`' is not followed by a number, then the symbols between the first pair of dollar signs *before* '`**`' are placed after the '`**`' which is left unchanged.

The symbol '`&`' causes the next symbol to be ignored as substitution rules are applied. The turtle is not affected by '`&`'. The symbol '``' (grave) causes the turtle to ignore the next symbol. The application of substitution rules is not affected by '``'.

The first '`#`' causes substitution rules not to be applied until the second '`#`'. The third '`#`' causes substitution rules not to be applied until the fourth '`#`'. The pattern continues. The turtle is not affected by '`#`'. Also, '`#`' and '`&`' do not affect the counting of '`$`' for asterisk commands. However, '`#`' and '`&`' can affect each other. For '`#FW#FX#FY#FZ#`', substitution rules will be applied to '`FX`' and '`FZ`' but not '`FW`' or '`FY`'. For '`&#FW#FX#FY#FZ#`', the situation is reversed.

10) Extra Line Segments

```
HeptD {~
  Angle 7
  Axiom ["K]"H
  K=F+F+"K
  H=G+G+G+"H
}
```



At times it is useful to connect points that are not consecutive points in the path of the turtle. '`HeptD`' uses '`"`' to draw diagonals in a heptagon. The command string for **Order 7** begins with '`["F+F+"F+F+"F+F+"F+F+"F+F+"F+F+"F+F+"K]`'. Do not confuse '`"`' with '`'`'. When the turtle sees the first '`"`', the turtle remembers its location which is vertex 0 of the heptagon. The next time the turtle sees '`"`' it is at vertex 2. It draws a line connecting vertex 0 and vertex 2. It also remembers its current location. Then when it comes to the next '`"`' it is at vertex 4. So it draws a line connecting vertex 2 and vertex 4. It also remembers its current location. Eventually the turtle comes to '`]`' which restores it to its condition at '`[`' where it had not yet remembered any locations.

The commands continue with '`"G+G+G+"G+G+G+"G+…`'. The first '`"`' tells the turtle to remember its current location at vertex 0. It sees the next '`"`' when it is at vertex 3. So it draws a line between vertex 0 and vertex 3. It also remembers its location at vertex 3. Then when it comes to vertex 6 it sees the next '`"`' and draws a line between vertex 3 and vertex 6. The turtle continues until it has drawn all the diagonals.

Order 7 of '`HeptD`' draws each edge of the heptagon twice since '`F`' occurs 14 times in the command string. **Order 7** of '`HeptD1`' draws each edge only once. However, the turtle crawls around the heptagon 6 times instead of 5 times.

```
HeptD1 {~
  Angle 7
  Axiom [A]["K]"H
  A=F+A
  K=G+G+"K
  H=G+G+G+"H
}
```

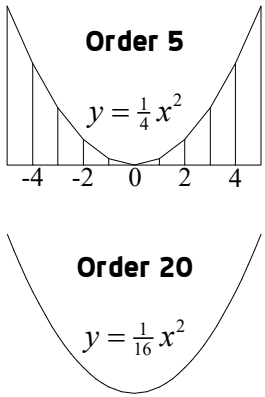
When the turtle is not between a pair of brackets, it is at nesting level 0. When it is between a single pair of brackets, it is at nesting level 1. When it is between two pairs of brackets, it is at nesting level 2, etc.

If the turtle comes to '`[`' after '`"`', it stores on the stack a copy of its memory that it made at the '`"`'. Then if it comes to '`"`', it draws a line segment from the remembered location to the current location. When it comes to '`]`', it restores the memory it had when it came to the '`[`', unless '`".`' (see next page) is between '`[`' and '`]`'.

If the turtle is at nesting level 1 when it sees '`".`', it draws a line between the current location and the nesting level 0 location memory that is on the stack. Also, the nesting level 0 memory on the stack is set to the current location.

```
Parabola {~
  Angle 4
  Axiom "[X]|!X
  X=XF[@I4+FYYYZ".]
  Y=FY
  Z=YYZ
}

ParabolaS {~
  Angle 4
  Axiom "[X]|!X
  X=XG[@I16+GYYYZ".]
  Y=GY
  Z=YYZ
}
```



'`"[XF[@I4+FYYYZ".]F[@I4+FFYFYFYYYZ".]`' is the first part of the command string for **Order 5** of '`Parabola`'. The first '`"`' stores (0, 0) as the nesting level 0 location memory. The first '`[`' puts that memory on the stack and also makes that the nesting level 1 location memory. Then '`F`' has the turtle crawl to (1, 0). Then '`[`' puts (0, 0) on the stack as the nesting level 1 memory and also makes that the nesting level 2 memory.

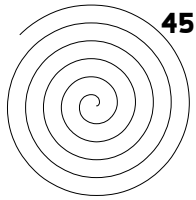
Then '`@I4+F`' has the turtle crawl to (1, 0.25). The turtle is currently in nesting level 2. So when it comes to '`".`', it looks on the stack at the nesting level 1 memory and draws a line between that location and the current location. It also makes the current location be the nesting level 1 memory. So it draws a line between (0, 0) and (1, 0.25). It also makes (1, 0.25) be the nesting level 1 memory on the stack.

When the turtle comes to '`]`', it returns to (1, 0) and to nesting level 1. So its location memory is now (1, 0.25). When it sees '`F`', it crawls from (1, 0) to (2, 0). Then '`[`' puts it in nesting level 2 and has it store (1, 0.25) on the stack as the nesting level 1 location memory. Then '`@I4+FFYFYF`' has it crawl to (2, 1). So when it sees '`".`', it draws a line from (1, 0.25) to (2, 1) and stores (2, 1) on the stack as the nesting level 1 location. The nesting level 2 location is not changed. Then at '`]`' the nesting level 2 location is forgotten, and the turtle returns to nesting level 1 as it continues its journey.

For **Order 5** of '`Parabola`', the rightmost point of the parabola is at (5, 6.25). For **Order 20** of '`ParabolaS`', the rightmost point of the parabola is at (20, 25). These L-systems use the single symbol '`"`' which should not be confused with '`''`' (two apostrophes).

11) Simple Arithmetic

```
Archimedes {~
  Angle 8
  Axiom '$1$B
  A=@=
  B=LA*1B
}
```



For ‘Archimedes’, **Order 0** is ‘'1B’ and **Order 1** is ‘'2LA*1B’. The ‘'’ (apostrophe) resulted in ‘1’ increasing to ‘2’. **Order 2** is ‘'3L@=2LA*1B’, and **Order 3** is ‘'4L@=2L@=3LA*1B’. The first number after ‘'’ is increased by 1 as the next order is made. The **Order 45** command string begins with ‘'46L@=2L@=3L@=4L@=5L’.

The turtle starts at the center of the spiral. The first symbol the turtle understands is ‘L’ which tells it to draw an arc while turning to the left. The arc is $\frac{1}{8}$ of a circle because of ‘Angle 8’. The radius of the circle is 1 unit since that is the initial crawl length.

Then the turtle sees ‘@=2’. This is an *absolute command* that *sets* the crawl length to 2 instead of *multiplying* it by 2. Then ‘L’ has the turtle draw an arc of a circle whose radius is 2.

If the turtle would see ‘@=8@@’, then ‘@=8’ would tell it to set its crawl length to 8. The ‘@@’ would tell it to multiply that crawl length by the previous multiplier which is viewed as 8 even though ‘@=8’ is an absolute command.

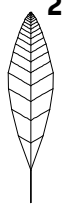
Then the turtle sees ‘@=3’ which tells it to set its crawl length to 3. Then ‘L’ tells the turtle to draw an arc of a circle whose radius is 3. The turtle keeps crawling until it obeys the last ‘L’ in the command string which ends with ‘@=43L@=44L@=45LA*1B’.

When the computer makes **Order 46**, it replaces ‘*1’ with the symbols between the first two dollar signs of **Order 45**. So ‘*1’ is replaced with ‘46’. Also, the beginning of the command string is changed from ‘'46’ to ‘'47’ since ‘'’ tells the computer to increase the next number in the string by 1.

The results of ‘:’ and ‘'’ should not be confused. For ‘:’ to affect a number, the number must *immediately* follow it. This is in contrast to ‘'’. Symbols may come between ‘'’ and the number to be increased. Those symbols are ignored as substitution rules are applied. However, if ‘*1’ or something similar is processed, then a ‘$’ may be counted even if it is between ‘'’ and the following number. In the above ‘Archimedes’ command strings, the first ‘$’ is counted even though it is between ‘'’ and the following number.

Order 21 of ‘Leaf124e’ makes a design like one on p. 124 of *The Algorithmic Beauty of Plants*. ‘Leaf124e’ uses both ‘'’ and ‘"’ which shouldn’t be confused. When a number is immediately after ‘'’ (two apostrophes), then that number is added to the next number that appears in the command string.

```
Leaf124e {~
  Angle 12
  Axiom %Z$".H$%+++"[A]"
  A=U[--K*1VN*2N][A][++K*1VN*2N]
  B=B@=4YF
  H=.H
  K=:
  N=#
  U=@=5XF
  V=B
  X=@1.2X
  Y=@1.1Y
  Z=''4$0$
}
```



The command string for **Order 1** starts with the sequence ‘%''4$0$$"..H$%’. For **Order 2**, that section changes to ‘%''4$4$$"...H$%’. Then for **Order 3** that section becomes ‘%''4$8$$"....H$%’. When **Order 4** is made, that section is ‘%''4$12$$".....H$%’. The second number increases by 4 each time since ‘''4’ precedes that number.

In the portion after the second ‘%’, **Order 2** has the sequence ‘#"..H#’ twice (this isn’t shown). When **Order 3** is made, the first ‘#’ results in substitution rules not being applied until after the second ‘#’. Then substitution rules are applied until the third ‘#’. Then substitution rules are not applied until after the fourth ‘#’. This keeps the ‘H’ substitution rule from being applied in those sections. The ‘#’ does not keep the turtle from reading symbols.

Numbers that may be used with ‘'’ are whole numbers such as 0, 1, 2, 3, etc. Computers use *binary numbers* which are written using the digits 0 and 1. A binary digit is a *bit*. If 3 bits are used, 8 numbers are possible which can be viewed as *signed* or *unsigned* as shown in the table below. The table shows that 111 can stand for 7 or -1. For emphasis, the table uses “+” which often isn’t shown.

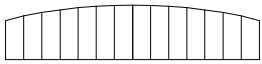
	000	001	010	011	100	101	110	111
Unsigned	0	1	2	3	4	5	6	7
Signed	0	+1	+2	+3	-4	-3	-2	-1

If 32 bits are used, 1111 1111 1111 1111 1111 1111 1111 1111 can stand for 4,294,967,295 or -1. Also, unsigned 4,294,967,294 is equivalent to signed -2, and 4,294,967,293 is equivalent to -3.

A number after ‘''’ uses 32 bits and must be unsigned. So -1 is written as ‘4294967295’ in ‘EllipseRoof’. The chart below shows the initial symbols all of the way through ‘K’ for each specified order.

```
EllipseRoof {~
  Angle 4
  Axiom ''4294967295$0$:1&~$100$&~KV
  B=''
  H=@.3+F".
  K=&~B*1$*2$&~:1&K
  N=F[@=Q*2H]:1&N
  Z=[@=10H]:1&N
  V=[Z]|!Z
}
```

Order 17



0	''4294967295$0$:1&~100&~K
1	''4294967295$4294967295$:&~100&~&~B*1$*2$&~:1&K
2	''4294967295$4294967294$~&~''4294967295$100$&~:&K
3	''4294967295$4294967293$~''4294967295$99$&~&~B*1$*2$&~:1&K
4	''4294967295$4294967292$~&~''4294967293$99$&~:&K
5	''4294967295$4294967291$~''4294967293$96$&~&~B*1$*2$&~:1&K
6	''4294967295$4294967290$~&~''4294967291$96$&~:&K
7	''4294967295$4294967289$~''4294967291$91$&~&~B*1$*2$&~:1&K

The second number in the string decreases by 1 as the order increases (even when going from ‘0’ to ‘4294967295’ which is equivalent to -1). This is because the first number in the string is added to it, where that first number is -1 written as ‘4294967295’. **Order 6** has ‘''4294967291$96$’ where ‘4294967291’ is equivalent to -5. Thus ‘''4294967291$96$’ changes to ‘''4294967291$91$’ in the next order.

‘…4294967290…96…[@=Q*2H]…’ is a condensed form of the **Order 6** command string. ‘[@=Q*2H]’ becomes ‘[@=Q96@.3+F".]’ in the next order. The absolute command ‘@=Q96’ sets the crawl length to $\sqrt{96}$. Then ‘@.3’ multiplies that length by 0.3 so the resulting crawl length is $\frac{3}{10}\sqrt{96}$. That is how far the elliptical roof is above the *x*-axis when $x = 2$. The equation for the elliptical roof can be written as $y = \frac{3}{10}\sqrt{10^2 - x^2}$. The ‘EllipseRoof’ L-system is designed so that numbers after ‘Q’ in the command string are of the form $10^2 - x^2$ up through **Order 23**. At that order, the numbers after ‘Q’ are 99, 96, 91, 84, 75, 64, 51, 36, 19, and 0. These come from *x* values that are natural numbers from 1 through 10.

12) Command Summary

Turtle commands for regular L-systems

	Example	Explanation
'FG' commands	F	Crawl forward and draw a line segment.
	G	Go forward without drawing.
	+	Turn counterclockwise.
	-	Turn clockwise.
	|	Turn counterclockwise 180° if that is allowed; otherwise keep rotating until an allowed angle.
'DM' commands	D	Crawl forward and draw a line segment.
	M	Move forward without drawing.
	\22.5	Turn counterclockwise 22.5°.
	/15	Turn clockwise 15°.
General commands	!	Reverse meanings of '+', '-', '\', and '/'.
	@.7	Multiply the crawl length by 0.7.
	@I3	Multiply the crawl length by the inverse of 3.
	@Q2	Multiply the crawl length by the square root of 2.
	@IQ8	Multiply the crawl length by the inverse of the square root of 8.
	C15	Set color number to 15. This is the default color.
	<2	Increase color number by 2. If the result is 256 or 257, then change to 1 or 2 respectively.
	>1	Decrease color number by 1. If the result is 0, then change the number to 255.
	[	Push current condition onto top of stack.
	]	Pop top condition off of stack and make that condition the current condition.

Do not use commas when writing numbers in L-systems.

Each regular L-system needs an ‘Angle’ command. This command affects ‘FG’ commands but not the ‘DM’ commands shown on the previous page. The turtle keeps track of two angles, one for the ‘FG’ commands and one for the ‘DM’ commands.

Substitution rules for regular L-systems

Example	Explanation
X=FX-FY+FX	Replace ‘X’ with ‘FX-FY+FX’.
F=	Replace ‘F’ with nothing. That is, delete ‘F’.
G=GF-FG+ G=GF-FF	Replace ‘G’ with ‘GF-FG+GF-FF’. A substitution rule may be on multiple lines like this.

Additional turtle commands available in arc L-systems

	Example	Explanation
‘FG’	L	Draw an arc while turning to the left.
	R	Draw an arc while turning to the right.
‘FG’ & ‘DM’	=	Set the ‘DM’ angle equal to the ‘FG’ angle.
	=I	Do the inverse of ‘=’ if possible. That is, set the ‘FG’ angle equal to the ‘DM’ angle. If that can’t be done, then set the ‘FG’ angle to the allowed angle that is closest to the ‘DM’ angle. If the two closest angles are equally close, then pick one of them (‘!’ reverses which angle is picked).
‘DM’	(10	Draw an arc while turning to the left 10°.
	)2.5	Draw an arc while turning to the right 2.5°.
	\=90	Turn until the direction is 90°.
	/=45	Turn until the direction is -45°.
	(=120	Draw an arc while turning to the left until the direction is 120°.
	)=90	Draw an arc while turning to the right until the direction is -90°.

'DM'	\?7.38	Pick an angle smaller than 7.38° and turn to the left that amount.
	/?15	Pick an angle smaller than 15° and turn to the right that amount.
	(?8.3	Pick an arc measure smaller than 8.3° and draw an arc while turning that amount to the left.
	)?.5	Pick an arc measure smaller than 0.5° and draw an arc while turning that amount to the right.
	\??2.1	Pick an angle smaller than 2.1° and turn to the left or to the right that amount.
	/??5.5	Pick an angle smaller than 5.5° and turn to the right or to the left that amount.
	(??7.25	Pick an arc measure smaller than 7.25° and draw an arc while turning to the left or to the right that amount.
	)??.8	Pick an arc measure smaller than 0.8° and draw an arc while turning to the right or to the left that amount. This is equivalent to '!(??.8'.
	\=?5	Pick an angle smaller than 5° and turn until that is the direction.
	/=?28.4	Pick an angle smaller than 28.4° and turn until the direction is the negative of that angle.
	\=??4	Pick an angle between -4° and 4°. Turn until that is the direction.
	/=??2.5	Pick an angle between -2.5° and 2.5°. Turn until that is the direction.
	(=?12	Pick an angle smaller than 12° and draw an arc while turning to the left until the direction is that angle.
	)=?9.2	Pick an angle smaller than 9.2° and draw an arc while turning to the right until the direction is the negative of that angle.

'DM'	(=??8	Pick an angle between -8° and 8° and draw an arc while turning to the left or to the right until the direction is that angle.
	)=??1.9	Pick an angle between -1.9° and 1.9° and draw an arc while turning to the right or to the left until the direction is that angle.
General commands	?54608	Change the seed for the pseudorandom number generator to 54,608.
	?	Ask the pseudorandom number generator for a number. This changes the state of the generator. The number is ignored.
	?.	Store the current state of the pseudorandom number generator in the most recent stack location.
	?..	Store the current state of the pseudorandom number generator in the next to most recent stack location.
	@?2	Pick a number from 1 up to (but not including) 2 and multiply the crawl length by that number.
	@?.5	Pick a number from 1 down to (but not including) 0.5 and multiply the crawl length by that number.
	@??3	Pick a number from 1 up to (but not including) 3 and multiply the crawl length by that number or its inverse.
	@??.73	Pick a number from 1 down to (but not including) 0.73 and multiply the crawl length by that number or its inverse.
	@I@	Multiply the crawl length by the inverse of the previous multiplier.
	@@	Multiply the crawl length by the previous multiplier.
	@=3	Set the crawl length to 3.
	@=Q3	Set the crawl length to the square root of 3.

General commands	@=?4.5	Pick a number from 1 up to (but not including) 4.5 and set the crawl length to that number.
	@=??.8	Pick a number from 1 down to (but not including) 0.8 and set the crawl length to that number or its inverse.
	!	Reverse the meanings of ‘L’, ‘R’, ‘(’, and ‘)’ as well as ‘+’, ‘-’, ‘\’, and ‘/’.
	`G	Ignore ‘G’.
	%F+GF%	Ignore ‘F+GF’.
	"	Remember the current location and draw a line segment to the previous remembered location (if there was one).
	".	Store the current location at the top of the stack and draw a line segment to the previous remembered location that was there on the stack (if there was such a memory). If the current nesting level is 0, then ‘".’ is treated as ‘"’.
	"..	Store the current location in the next to the top stack space and draw a line segment to the previous remembered location that was there on the stack (if there was such a memory). If the current nesting level is 0, then ‘"..’ is treated as ‘"’. If the current nesting level is 1, then ‘"..’ is treated as ‘".’.
	""	Draw a line segment from a remembered location (if there was one) and then clear the location memory.
	"".	Draw a line segment from a remembered location that was stored at the top of the stack (if there was such a memory) and then clear that memory on the stack. If the current nesting level is 0, then ‘"".’ is treated as ‘""’.

When the crawl length is set by an absolute command, that length is then viewed as the previous multiplier.

Additional substitution rules available in arc L-systems

Example	Explanation
`E=,3,1` `E=F,G`	Around 75% of the time replace '`E`' with '`F`'. The rest of the time replace '`E`' with '`G`'.
`F=,7,3` `F=FF,`	Around 70% of the time replace '`F`' with '`FF`'. The rest of the time replace '`F`' with nothing (that is, delete '`F`').
`T=,0,1,1,1,1,1` `T=Z,1,3,5,7,9`	Pick an odd digit and replace '`T`' with that digit. The '`Z`' is picked 0% of the time. The reason for the '`0`' and the '`Z`' is because the first choice cannot begin with a digit.
`N=,3,1,1F,G,`	Around 60% of the time replace '`N`' with '`F`'. Around 20% of the time replace '`N`' with '`G`'. Around 20% of the time delete '`N`'.

Command string instructions for arc L-systems

Example	Explanation
`&F`	Ignore '`F`' when applying substitution rules.
`_F`	Ignore and delete '`F`'. The '`_`' is not deleted.
`~F+G-FF+F~`	Ignore and delete '`F+G-FF+F~`'. The first '~' isn't deleted.
`#F+G-FF+F#`	Ignore '`F+G-FF+F#`'.
`:4`	Change to '`:3`' since 4 – 1 = 3.
`:1`	Change to '`:`' which is equivalent to '`:0`'.
`:F`	Ignore and delete '`F`'. The '`:`' is also deleted.
`:0F`	Ignore and delete '`F`'. The '`:0`' is also deleted.
`'FH4`	Change to '`'FH5`' since 4 + 1 = 5. Do not apply substitution rules to '`FH`'.
`''5K12`	Change to '`''5K17`' since 5 + 12 = 17. Do not apply a substitution rule to '`K`'.

`^`	Insert the entire command string after the ‘`^`’. That is, as an order is being produced from the previous order, insert the command string for that previous order after the ‘`^`’ (caret).
`*`	Find the first pair of dollar signs in the command string. Copy symbols between them. Place those symbols after ‘`*`’ (if any symbols were found).
`*1`	Find the first ‘`$`’ and the second ‘`$`’ in the command string and copy the symbols between them. If any such symbols exist, replace ‘`*1`’ with those symbols. If no such symbols exist, do not change ‘`*1`’.
`*2`	Find the second pair of dollar signs (the third and fourth ones) in the command string and copy the symbols between them. If any such symbols exist, replace ‘`*2`’ with those symbols. If there aren’t any such symbols, don’t change ‘`*2`’.
`**`	Search the symbols that precede ‘`**`’ for the first ‘`$`’ and the second ‘`$`’ before ‘`**`’. Copy the symbols between that first pair. If any such symbols exist, place them after ‘`**`’.
`**1`	Search for the first pair of dollar signs to precede ‘`**1`’. Copy the symbols between that first pair. If any such symbols exist, replace ‘`**1`’ with them. Otherwise, leave ‘`**1`’ alone.
`**2`	Search for the second pair of dollar signs to precede ‘`**2`’. Copy the symbols between that second pair. If any such symbols exist, replace ‘`**2`’ with them. Otherwise, leave ‘`**2`’ alone.

The symbols ‘`_`’ and ‘`:`’ may indirectly affect later symbols. For example, ‘`_&~F+F--F&~`’ becomes ‘`_~`’. The ‘`_`’ causes the first ‘`&`’ to be ignored (and deleted), so the first ‘`~`’ is not ignored.

For ‘`''$2$''4$3$_FG~+F~#FF#:3:F+@*1G+@I**1+&F:1^`’, if ‘`F=FF`’ is the only substitution rule, then the next command string is ‘`''$3$''4$7$_G~#FF#:2+@2G+@I3+&F:^'$2$''4$3$_FG~+F~#FF#:3:F+@*1G+@I**1+&F:1^`’.

```
RectBorderAV {~ ;Order 3 is the border on the front cover.
  Angle 4                 ;A semicolon indicates a comment
  Axiom YYKYKYYKYK        ;which is ignored by the computer
  K=FRFFRFR@I2FRRLLF@2LFL ;as it reads the L-system.
  Y=FRFFRFR@I2FRRLLF@2LFLFFLFY
}

DendriteS {~55977734 ;Order 11 is the design above the
  Angle 6             ;title on the front cover.
  Axiom %$B$%[A][!A]+[A][!A]+[A][!A]+[A][!A]+[A][!A]+[A]!A
  A=:2:1&~*1&~A ;The design under the title has 6117896 for
  B=BF[E]       ;the seed (instead of 55977734).
  E=,2,1,1,1,1,1,1
  E=[B]+[B][!B]+[B]!B,+[B][!B]+[B]!B,+[B]!B,
  E=++[B]!B,[B]+[B]!B,[B]++[B]!B,B
  F=@5@I3F@3@I5
}

Florets14 {~ ;Order 2800 is the central design near the
  Axiom K    ;bottom of page 3. It has 2800 small circles.
  K=K\137.507764''14@Q1M@I@(360
}

PlantTilt9 {~    ;Order 55 is near the bottom of page 3.
  Axiom \90DA    ;It is on the right. The design on the
  D=@1.11D@I1.11 ;left is a mirror image of this design.
  A=[@1.2:9&S]\2DB ;This arc L-system doesn't have F, G,
  B=[!:9&S]\2DA    ;L, R, +, -, or |. So it doesn't need
  S=/45DA          ;an Angle command.
}

TriSpiral7A {~ ;Order 26 is the design on the back cover.
  Angle 21
  Axiom +[LX]=\120=I[LX]\120=ILX
  X=[@.2YYY]HH[!@.22YYY]HHX
  Y=HY
  H=@.96L
}

DoubleSpiral10 {~ ;Order 30 is the design to the right of
  Angle 10        ;the page number below. The design to the
  Axiom --[LA]|LA ;left is a mirror image of this design.
  A=@.9LA
}
```

The axiom for '`TriSpiral7A`' has the commands '`=`' and '`=I`'. These commands, used with '`\120`', simplify experimenting with different '`Angle`' numbers.