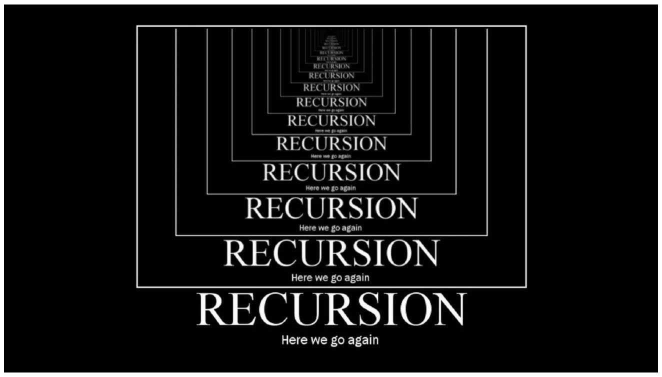
Asian MathSci League, Inc (AMSLI)
Partner: National Olympiad in Informatics Philippines (NOI.PH)
Email address: amsliphil@yahoo.com / ask@noi.ph
Contact Nos: +632-9254526 +63906-1186067

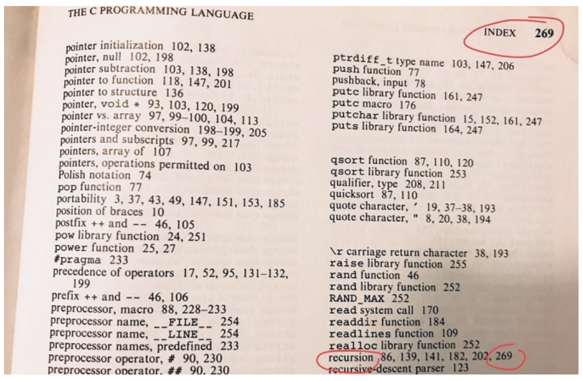
Student Copy
AIEP Scratch 2 Session 4B

INTRODUCTION TO RECURSION

"To understand recursion, one must first understand recursion."
~ Stephen Hawking

In computer science, RECURSION is defining a task in terms of the task itself (to be more precise, in terms of smaller versions of the task). For instance, a RECURSIVE PROCEDURE is a procedure that calls itself. While it may be difficult to grasp this idea at first, these memes and Easter eggs may help you visualize what recursion is:




THE C PROGRAMMING LANGUAGE

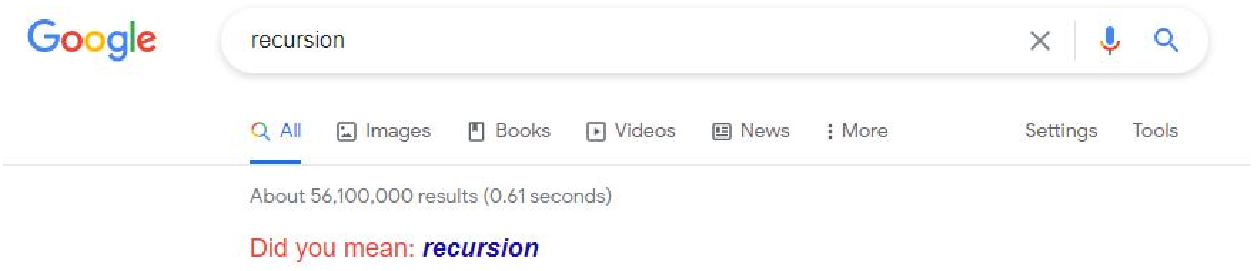
INDEX 269

pointer initialization 102, 138
pointer, null 102, 198
pointer subtraction 103, 138, 198
pointer to function 118, 147, 201
pointer to structure 136
pointer, void * 93, 103, 120, 199
pointer vs. array 97, 99–100, 104, 113
pointer-integer conversion 198–199, 205
pointers and subscripts 97, 99, 217
pointers, array of 107
pointers, operations permitted on 103
Polish notation 74
pop function 77
portability 3, 37, 43, 49, 147, 151, 153, 185
position of braces 10
postfix ++ and -- 46, 105
pow library function 24, 251
power function 25, 27
#pragma 233
precedence of operators 17, 52, 95, 131–132, 199
prefix ++ and -- 46, 106
preprocessor, macro 88, 228–233
preprocessor name, __FILE__ 254
preprocessor name, __LINE__ 254
preprocessor names, predefined 233
preprocessor operator, # 90, 230
preprocessor operator, ## 90, 230

ptrdiff_t type name 103, 147, 206
push function 77
pushback, input 78
putc library function 161, 247
putc macro 176
putchar library function 15, 152, 161, 247
puts library function 164, 247

qsort function 87, 110, 120
qsort library function 253
qualifier, type 208, 211
quicksort 87, 110
quote character, ' 19, 37–38, 193
quote character, " 8, 20, 38, 194

\r carriage return character 38, 193
raise library function 255
rand function 46
rand library function 252
RAND_MAX 252
read system call 170
readdir function 184
readlines function 109
realloc library function 252
recursion 86, 139, 141, 182, 202, 269
recursive-descent parser 123



recursion

About 56,100,000 results (0.61 seconds)

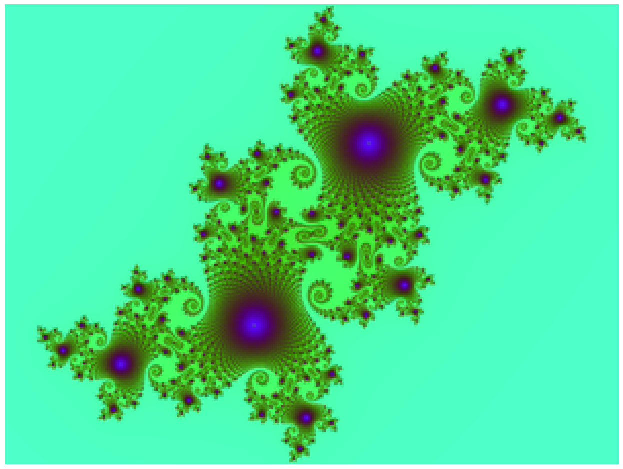
Did you mean: ***recursion***

Usually, programmers' first encounter with recursion is through mathematical concepts, like factorials and the Fibonacci sequence. However, Scratch's support for recursion is slightly less sophisticated compared to most other programming languages due to the fact that My Blocks (functions) do not return any values; they are merely procedures. The most common workaround is to store values inside a list.

*Prepared by **Mark Edward M. Gonzales***

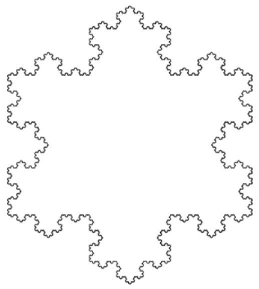
Therefore, this handout seeks to introduce you to the idea of recursion *the Scratch way*, that is, by taking a look at how recursion plays an important role in creating interesting geometric figures called FRACTALS.

A fractal is an infinitely complex shape that is usually SELF-SIMILAR. This means zooming in will produce the same (or roughly the same) figure as the original. Its history is closely tied to mathematics; the figure on the right shows a visualization of the Julia set (named after the French mathematician Gaston Julia). Meanwhile, the figure on the left shows a fractal occurring in nature: the Romanesco broccoli.



THE KOCH CURVE

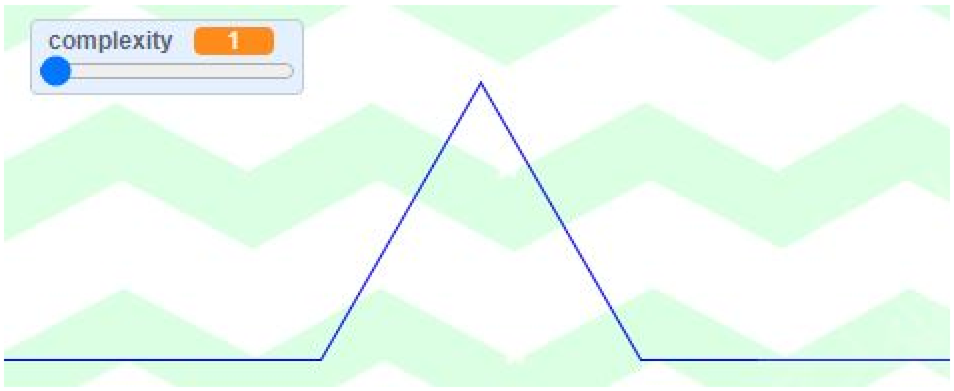
One of the first fractals to be formally described is the Koch snowflake, named after the Swedish mathematician Helge von Koch, who wrote a paper on it in 1904. Through this discussion, we will learn how to create half a Koch snowflake, which is called a Koch curve.



First, make sure that you have imported the Pen extension by clicking on the lower left button on the Scratch Editor. The preliminaries will be left to you as an exercise since our focus will be on how to develop and code the recursive procedure itself. All recursive procedures have two parts: a BASE CASE and a RECURSIVE CASE. We can think of the base case as the "simplest level" whereas the recursive case is the time when the procedure actually call itself.

Creating the Base Case

In coding the base case, we have to take a look at the simplest possible Koch curve, which is essentially a single spike:

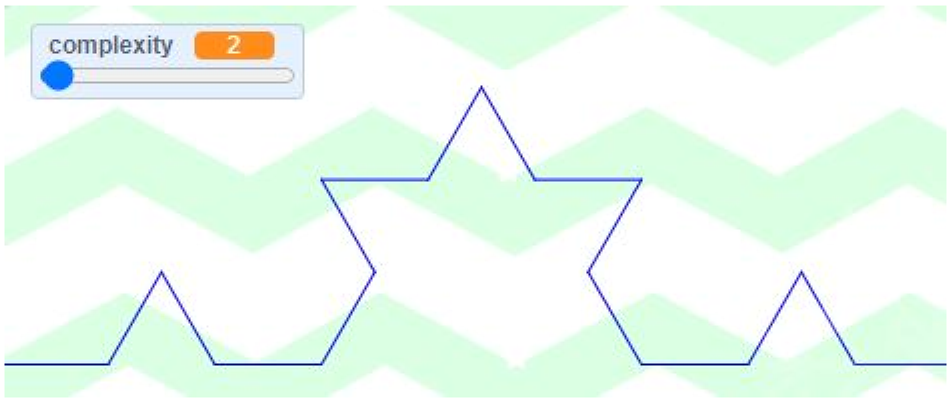


We have to draw four lines, rotating the sprite at each step as we change direction :

a. Draw the first horizontal line.
b. Rotate the sprite 60 degrees counterclockwise. *Why?*
c. Draw the diagonally ascending line.
d. Rotate the sprite 120 degrees clockwise. *Why?*
e. Draw the diagonally descending line.
f. Rotate the sprite 60 degrees counterclockwise. *Why?*
g. Draw the second horizontal line.

Creating the Recursive Case

In coding the recursive case, it is helpful to take a look at the second simplest Koch curve:



Notice that a spike has been added to each of the four segments from earlier. To be more precise, in order to form our "level 2" Koch curve, we have to create a "level 1" Koch curve on each of the segments in the "level 1" Koch curve. Observe our manner of definition: we used a smaller Koch curve to define a larger Koch curve. This is recursion.

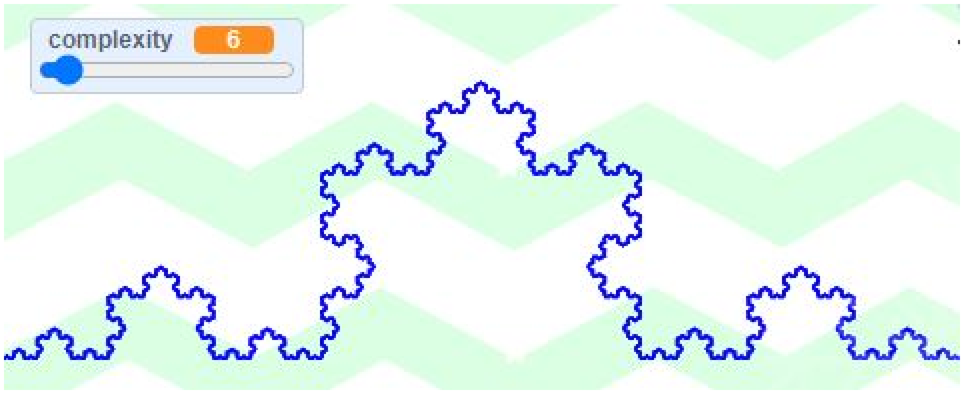
Therefore, our recursive case must involve four recursive calls:

a. Draw a "one-level-simpler" Koch curve.
b. Rotate the sprite 60 degrees counterclockwise. *Why?*
c. Draw a "one-level-simpler" Koch curve.
d. Rotate the sprite 120 degrees clockwise. *Why?*
e. Draw a "one-level-simpler" Koch curve.
f. Rotate the sprite 60 degrees counterclockwise. *Why?*
g. Draw a "one-level-simpler" Koch curve.

Combining the Base and Recursive Cases

If you paid particular attention to the memes found on the first page, this question might have crossed your mind: *How do a recursive procedure terminate if it just keeps on calling itself?* The answer lies in the base case[1], which is standalone.

The final My Block for creating the Koch curve has a conditional. If the complexity is equal to 1, then the base case is activated. Otherwise, the recursive case is activated. This act of combining the base and recursive cases is a pattern followed in all recursive procedures. Test if your code works by comparing it with the output below (the "level" is set to 6):



[1] As we progress with programming, we will learn that a recursion can have multiple base cases.

Forgetting to specify the base case is one of the most common programming bugs. The usual result is overloading the memory resources of the computer. This is referred to as STACK OVERFLOW[2]. Common indicators of this bug are:

a) Nothing is displayed on the screen (since the first/simplest step is missing).
b) The display on the screen does not stop (since it does not know when to end).
c) The program — if not your entire computer — hangs and crashes (as a result of overflowing a portion of your computer memory).

Note that, while recursion is an elegant method for creating programs, it is expensive in terms of space and time efficiency. As much as possible, if there exists a solution using `repeat` blocks, then it is better to just use `repeat` blocks. A more advanced alternative is to "memorize" recursion via a technique called dynamic programming.

ACTIVITY: KOCH SNOWFLAKE

Following the steps outline in this handout, create a Koch curve using the skeleton file Koch Curve_For Students.sb3.

You may need to turn on Turbo Mode to speed up the display for more complex curves. Experiment with the code, add some sprites and backgrounds, and be creative!

For the technical challenges:

a) Try to create a mirrored Koch curve (that is, it peaks downwards).
b) Try to create a whole Koch snowflake. You could use two sprites to do this. But, for a more authentic challenge on recursion, try doing this using only a single sprite.

`-- return 0; --`

[2] This is the name of the famous programming forum: https://stackoverflow.com/