Recursion in Mathematics and Programming

Summation

Compute summation

$$\sum_{k=1}^{n} = 1 + 2 + \ldots + n.$$

It's recursive definition:

$$\begin{aligned} \mathbf{sum}(0) &= 0 \\ \mathbf{sum}(n) &= \mathbf{sum}(n-1) + n \qquad (\text{for } n \geq 1) \end{aligned}$$

Source code "`sum.jl`".

Factorial

Factorial function

$$\mathbf{fact}(n) = n! = 1 \cdot 2 \cdot \ldots \cdot n.$$

It's recursive definition:

$$\begin{aligned} \mathbf{fact}(0) &= 1 \\ \mathbf{fact}(n) &= \mathbf{fact}(n-1) \cdot n \qquad (\text{for } n \geq 1) \end{aligned}$$

Source code "`fact.jl`".

Factorial

Evaluation of the expression **fact**(3)

$$
\begin{aligned}
\textbf{fact}(3) &\Rightarrow \textbf{fact}(2)\cdot 3\\
&\Rightarrow (\textbf{fact}(1)\cdot 2)\cdot 3\\
&\Rightarrow ((\textbf{fact}(0)\cdot 1)\cdot 2)\cdot 3\\
&\Rightarrow ((1\cdot 1)\cdot 2)\cdot 3\\
&\Rightarrow (1\cdot 2)\cdot 3\\
&\Rightarrow 2\cdot 3\\
&\Rightarrow 6
\end{aligned}
$$

Fibonacci numbers

Recursive definition of the Fibonacci numbers:

$$\begin{aligned}
&\mathbf{fib}(0) = 1 \\
&\mathbf{fib}(1) = 1 \\
&\mathbf{fib}(n) = \mathbf{fib}(n-1) + \mathbf{fib}(n-2) \qquad (\text{for } n \geq 2)
\end{aligned}$$

Source code “`fib.jl`”.

Fibonacci numbers

Evaluation of the expression $\mathbf{fib}(5)$

$$
\begin{aligned}
\mathbf{fib}(5) &\Rightarrow \mathbf{fib}(3) + \mathbf{fib}(4) \\
&\Rightarrow (\mathbf{fib}(1) + \mathbf{fib}(2)) + (\mathbf{fib}(2) + \mathbf{fib}(3)) \\
&\Rightarrow (1 + (\mathbf{fib}(0) + \mathbf{fib}(1))) + ((\mathbf{fib}(0) + \mathbf{fib}(1)) + (\mathbf{fib}(1) + \mathbf{fib}(2))) \\
&\Rightarrow (1 + (1 + 1)) + ((1 + 1) + (1 + (\mathbf{fib}(0) + \mathbf{fib}(1)))) \\
&\Rightarrow (1 + 2) + (2 + (1 + (1 + 1))) \\
&\Rightarrow 3 + (2 + (1 + 2)) \\
&\Rightarrow 3 + (2 + 3) \\
&\Rightarrow 3 + 5 \\
&\Rightarrow 8
\end{aligned}
$$

Exponential running time, $O(2^n)$, too slow to be practical.

Improved recursive Fibonacci

Evaluation of the expression **fib2**(5)

$$\begin{aligned}
\textbf{fib2}(5) &\Rightarrow \textbf{next}(1,1,2)\\
&\Rightarrow \textbf{next}(1,2,3)\\
&\Rightarrow \textbf{next}(2,3,4)\\
&\Rightarrow \textbf{next}(3,5,5)\\
&\Rightarrow 3+5\\
&\Rightarrow 8
\end{aligned}$$

Linear time, $O(n)$, this is a real improvement.

A simple calculator language

Syntax

$$
\begin{aligned}
< expr > \quad ::= \quad & \textit{strings of digits}, \text{ represent integer numbers} \\
| \quad & (\ < expr > \ + \ < expr > \ ) \\
| \quad & (\ < expr > \ - \ < expr > \ ) \\
| \quad & (\ < expr > \ \times \ < expr > \ )
\end{aligned}
$$

Examples:

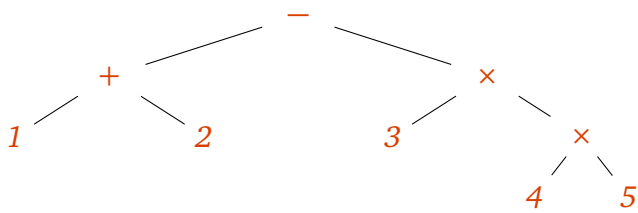
$$55$$

$$(\ 1 \ + \ 2 \ )$$

$$(\ (\ 1 \ + \ 2 \ ) \ - \ (\ 3 \ \times \ (\ 4 \ \times \ 5 \ ) \ ) \ )$$

A simple calculator language

$$(\ (\ 1 + 2 \ ) \ - \ (\ 3 \times \ (\ 4 \times 5 \ ) \ ) \ )$$



The evaluating function looks at the root of the tree only, the operators are evaluated one at a time

$$eval\big[\ (\ E_1 \ - \ E_2 \ ) \ \big]$$

$$\Rightarrow \quad eval\big[E_1\big] - eval\big[E_2\big]$$

A simple calculator language

Define the evaluation function:

Operators

$$eval\left[\ (\ E_1 + E_2\ )\ \right] \quad = \quad eval\left[E_1\right] + eval\left[E_2\right]$$

$$eval\left[\ (\ E_1 - E_2\ )\ \right] \quad = \quad eval\left[E_1\right] - eval\left[E_2\right]$$

$$eval\left[\ (\ E_1 \times E_2\ )\ \right] \quad = \quad eval\left[E_1\right] \times eval\left[E_2\right]$$

Numbers

$$eval\left[n\right] \quad = \quad n$$

A simple calculator language

$$
\begin{aligned}
& eval\big[\ (\ (\ 1 + 2\ ) - (\ 3 \times (\ 4 \times 5\ )\ )\ )\ \big] \\
\Rightarrow\ & eval\big[\ (\ 1 + 2\ )\ \big] - eval\big[\ (\ 3 \times (\ 4 \times 5\ )\ )\ \big] \\
\Rightarrow\ & (eval\big[1\big] + eval\big[2\big]) - (eval\big[3\big] \times eval\big[\ (\ 4 \times 5\ )\ \big]) \\
\Rightarrow\ & (1 + 2) - (3 \times (eval\big[4\big] \times eval\big[5\big])) \\
\Rightarrow\ & 3 - (3 \times (4 \times 5)) \\
\Rightarrow\ & 3 - (3 \times 20) \\
\Rightarrow\ & 3 - 60 \\
\Rightarrow\ & -57
\end{aligned}
$$

Recursion in Mathematics

Some prominent examples of recursion

- *Euclid's algorithm*
  a algorithm for computing the greatest common divisor.
- *Newton's method*
  a recursive method for finding successively better approximations to the roots of a real-valued function.
- Complex objects such as fractals can be defined using recursive definition.

Define set M

$c \in M$ if the sequence

$$z_0 = 0$$
$$z_n = z_{n-1}^2 + c \quad \text{does not go to infinity.}$$

Define set M

$c \in M$ if the sequence

$$z_0 = 0$$
$$z_n = z_{n-1}^2 + c \quad \text{does not go to infinity.}$$

Numbers c and z_n are *complex numbers*. So, we need to quickly learn how to add and multiply them.

Quick intro to complex numbers

The set of complex numbers $\mathbb{C}$ is an extension of the set of real numbers $\mathbb{R}$.

Any complex number can be represented by a pair of real numbers

$$(x, y) \qquad x, y \in \mathbb{R}$$

x is the *real part*, and y is the *imaginary part*.

Alternative notation. We can represent the pair as a sum of its real and imaginary part

$$(x, y) = x + yi$$

$i = \sqrt{-1}$ is the *imaginary unit*.

Quick intro to complex numbers

Addition

$$(a, b) + (c, d) = (a + c, b + d)$$

Multiplication

$$(a, b) \cdot (c, d) = (ac - bd, bc + ad)$$

Particularly,

$$i^2 = i \cdot i = (0, 1) \cdot (0, 1) = (0 - 1, 0 + 0) = (-1, 0) = -1$$

Define set M

A complex number $c \in M$, if the sequence

$$z_0 = 0$$
$$z_n = z_{n-1}^2 + c \quad \text{does not go to infinity.}$$

$z_n = x_n + y_n i$ will go to infinity if one of its components gets large:

$$\sqrt{x_n^2 + y_n^2} \geq 2$$

This condition is used practically to compute the membership of c in the set.

Source code "`mset.jl`".

L-systems

An L-system or Lindenmayer system is a parallel *rewriting system*.

It consists of

- an alphabet of symbols
- an initial string (called an axiom) to start construction
- a collection of production rules that expand each symbol into some larger string of symbols

Example:

Alphabet:	$\{A, B\}$
Initial string:	A
Production rules:	$A \rightarrow AB$
	$B \rightarrow A$

L-systems

Example:

Alphabet:	$\{A, B\}$
Initial string:	A
Production rules:	$A \rightarrow AB$
	$B \rightarrow A$

Start rewriting:

$$n = 0 : A$$
$$n = 1 : AB$$
$$n = 2 : ABA$$
$$n = 3 : ABAAB$$
$$n = 4 : ABAABABA$$
$$n = 5 : ABAABABAABAAB$$
$$n = 6 : ABAABABAABAABABAABABA$$

L-systems

Koch curve:

Alphabet:	$\{F, L, R\}$
Initial string:	F
Production rules:	$F \to F\ L\ F\ R\ F\ R\ F\ L\ F$
	$L \to L$
	$R \to R$

When F stands for moving forward, and L and R are the commands for turning left and right

Source code "`lsys.jl`".