Recursion

January 18, 2026

1 Recursion

1.1 External Resources

- YouTube Video - https://youtu.be/VvVlYK8pJl8
- YouTube Podcast - https://youtu.be/p04J7-oSnDQ
- NotebookLM learning materials - https://notebooklm.google.com/notebook/37a704cf-ca5c-4394-999f-f5f4831b72e6

1.2 Overview

- defining something in terms of itself usually at some smaller scale, perhaps multiple times, to achieve your objective
  - e.g., a human being is someone whose mother is a human being
  - directory is a structure that holds files and (smaller) directories, etc.
  - fractals is a drawing which also has self-similar structure, where it can be defined in terms of itself
- in programming, functions can generally call themselves to solve similar but smaller subproblems
  - this technique is called recursion

1.3 Definitions

- **Recursion:** The process of solving a problem by reducing it to smaller versions of itself
- **Recursive definition:** a definition in which something is defined in terms of smaller version of itself
- **Recursive algorithm:** an algorithm that finds a solution to a given problem by reducing the problem to smaller versions of itself
- **Infinite recursion:** recursive call that never stops

1.3.1 general construct of recursive algorithms

- recursive algorithms/solutions have base case(s) and general case(s):

1. must have one or more base case(s)
   - provides direct answer tha t makes the recursion stop
   - answer to the smallest subproblem
2. must have one or more general case(s)
   - functions recursively reduce themselves to one of the base case(s)

[1]:
```
#include <iostream>
#include <thread>
#include <chrono>
#include <cassert>

using namespace std;
```

[2]:
```
// Recursively print countdown from 10-1 and blast off!
void countDown(int n)
{
    if (n == 0) { // base case
        cout << "Blast Off!" << endl;
        this_thread::sleep_for(chrono::seconds(1));
    } else { // recursive/general case
        cout << n << endl;
        this_thread::sleep_for(chrono::seconds(1));
        countDown(n-1);
        //cout << n << endl;
      }
}
```

[3]:
```
countDown(10);
```

```
10
9
8
7
6
5
4
3
2
1
Blast Off!
```

1.3.2 Sum of First N positive integers

- use recursion to find the sum of first N positive integers

```
first_sum(1) = 1  (base case)
first_sum(n) = n + first_sum(n-1) for n > 1 (general case)
```

[4]:
```
long first_sum(int n)
{
    if (n == 1) { // base case
        return 1;
    } else { // general case
        return n + first_sum(n-1);
    }
}
```

[5]:
```
cout << "Sum of first 10 positive integers is: " << first_sum(100) << endl;
```

```
Sum of first 10 positive integers is: 5050
```

1.4 Fibonacci numbers

- Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …
- devised by Fibonacci (1170-1250), who used the sequence to model the breeding of (pairs) of rabbits
- say in generation 7 you had 21 pairs in total, of which 13 were adults, then in next generation the adults will have bred new children, and the previous children will have grown up to become adults. So, in generation 8, you'll have 13+21=34 rabbits, of which 21 are adults.

1.4.1 Fibonacci number definition

```
fib(0) = 0 (base case 1)
fib(1) = 1  (base case 2)
fib(n) = fib(n-1) + fib(n-2) for n >= 2  (general case)
```

[6]:
```
// Finding Fibonacci number in series
int fib_count = 0;
```

[7]:
```
long fib(unsigned int n)
{
    fib_count++;
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    long f = fib(n-1) + fib(n-2);
    return f;
}
```

[8]:
```
cout << fib(32) << endl;
```

```
2178309
```

[10]:
```
cout << "Fibonacci function was called " << fib_count << " times." << endl;
```

```
Fibonacci function was called 7049155 times.
```

1.4.2 visualize fib(4) using pythontutor.com

- https://pythontutor.com/render.html#code=%23include%20%3Ciostream%3E%0Ausing%20namespace%2
1%29%20%2B%20fib%28n-2%29%3B%0A%20%20%20%20return%20f%3B%0A%7D%0A%0Aint%20main%
frontend.js&py=cpp_g%2B%2B9.3.0&rawInputLstJSON=%5B%5D&textReferences=false

1.4.3 how many times is fib() called for fib(4)?

- Time complexity of recursive fib function is $O(2^n)$ or precisely $O(1.6180^n)$
  - 1.6180 is also called **golden ratio**

[2]:
```
size_t fib_count_tail = 0;
```

[ ]:
```
// Tail Recursive Fibonacci optimized version
long fib_tail(unsigned int n, long a, long b)
{
    fib_count_tail++;
    if (n == 0)
        return a;
    if (n == 1)
        return b;
    return fib_tail(n-1, b, a + b);
}
```

[6]:
```
cout << fib_tail(32, 0, 1) << endl;
```

```
2178309
```

[7]:
```
cout << "fib_tail function was called " << fib_count_tail << " times." << endl;
```

```
fib_tail function was called 32 times.
```

1.4.4 Factorial definition

```
1! = 1  (base case)
n! = n * (n-1)! for n > 1 (general case)
```

- Exercise - Implement factorial recursive solution; see homework assignment!

1.5 Exercises

1.5.1 GCD

Write a recursive function – gcd(a, b) – that finds the greatest common divisor of two given positive integers, a and b. - see algorithm in Khan Academy

1.5.2 Tower of Hanoi

Write a program that simulates the steps required to solve the "Tower of Hanoii" puzzle for some disks n. - https://www.mathsisfun.com/games/towerofhanoi.html

- Recursive algorithm
- If there are 1 or more disks to move:
  1. Move the top n-1 disks from needle 1 (source) to needle 2 (helper), using needle 3 (dest) as the intermediate needle
  2. Move disk number n from needle 1 (src) to needle 3 (dest)
  3. Move the top n - 1 disks from needle 2 (helper) to needle 3 (dest), using needle 1 (src) as the intermediate needle

[8]:
```
void moveDisks(int n, char src, char helper, char dst) {
    if (n > 0) {
        moveDisks(n-1, src, dst, helper);
        cout << "Move disk #" << n << " from " << src << " to " << dst << endl;
        moveDisks(n-1, helper, src, dst);
    }
}
```

[9]:
```
moveDisks(3, 'A', 'B', 'C');
```

```
Move disk #1 from A to C
Move disk #2 from A to B
Move disk #1 from C to B
Move disk #3 from A to C
Move disk #1 from B to A
Move disk #2 from B to C
Move disk #1 from A to C
```

1.6 Kattis problems

- the following Kattis problems can be solved using recursion:
- Last Factorial Digit: https://open.kattis.com/problems/lastfactorialdigit
- Watch Out For Those Hailstones! - https://open.kattis.com/problems/hailstone
- Out of Sorts - https://open.kattis.com/problems/outofsorts
  - Hint: implement recursive binary search and apply to the generated unsorted sequence