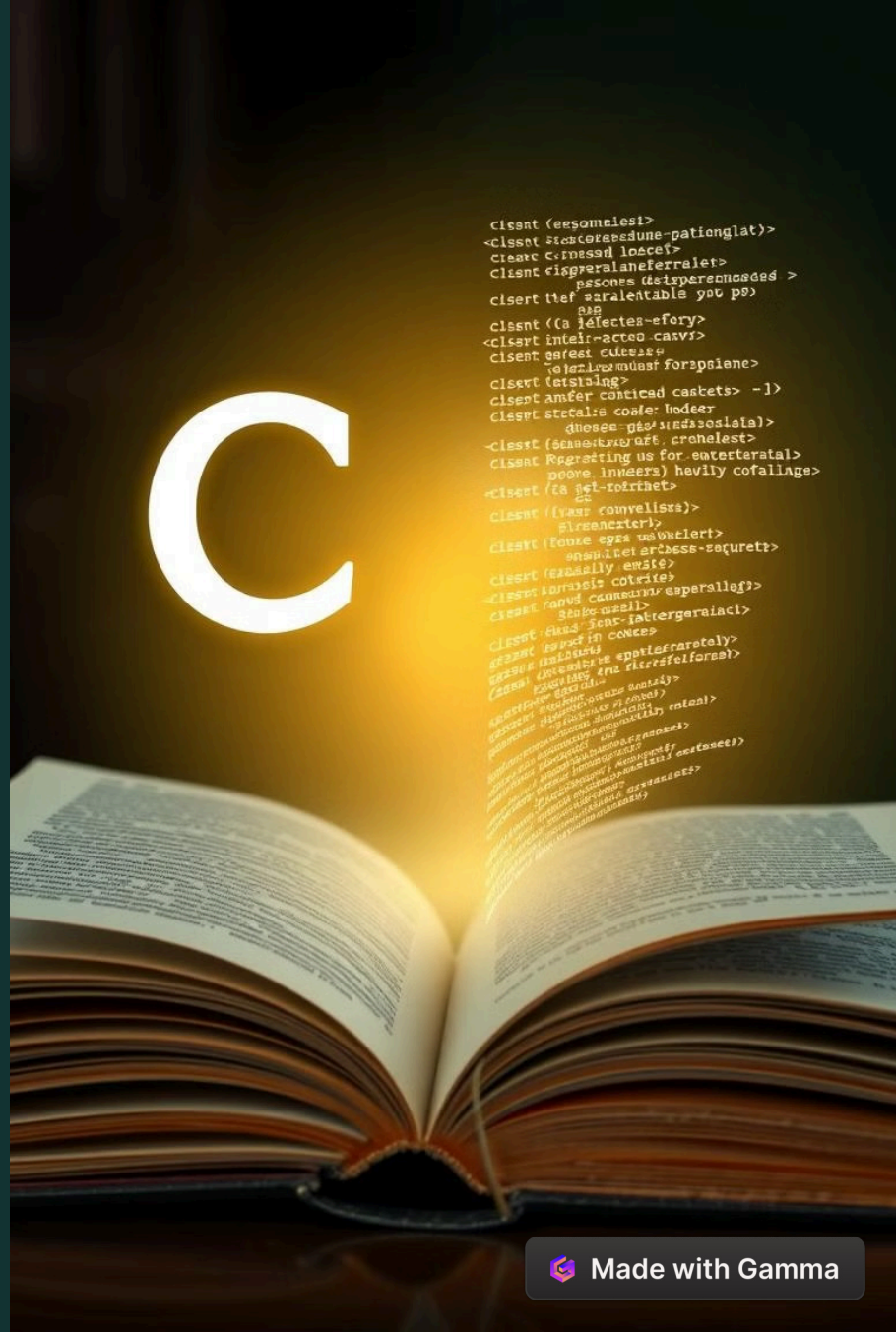
Functions and Recursion in C

We'll explore syntax, parameters, and scope. Learn to use functions for modularity and reusability. Finally, we will uncover the elegance of recursion.

by Hamdi Emad



Function Declaration and Definition

Most computer programs that solve real-world problems are much larger than the programs used in naive problems like printing something like a sentence, pattern or doing arithmetic/logic operations. Experience has shown that the best way to develop and maintain a large program is to construct it from smaller pieces, each of which is more manageable than the original program. This technique is called divide and conquer. Functions describe some key features of the C language that facilitate the design, implementation, operation and maintenance of large programs.

Declaration

Interface specification.

Example: `int multiply(int x, int y);`

Definition

Actual implementation.

Example: `int multiply(int x, int y) { return x * y; }`

Header Files

Organize declarations for reusability.

Function Parameters and Arguments

C functions have many types :-

(1) Library functions (printf, scanf, pow, sqrt, strlen, ...).
(2) User-defined functions (made by user for reusability principle and to perform specific tasks).
(3) Recursive functions (functions that call itself to reduce the time complexity and for solving problems).

Formal parameters are in the function definition. Arguments are values passed to the function. Pass by value creates a copy. Pass by reference modifies the original.

Pass by Value

Copies the argument's value.

Pass by Reference

Modifies the original variable using pointers.

Example: `void increment(int *n) { (*n)++; }`

Return Values and the `void` Type

There is another criteria for categorizing the functions based on the parameters and the return type.

1

1. Return Statement :-

Returns a value from a function if it's not a void function.

Example: `int square(int n) { return n * n; }`

2. `void` Type :-

Function returns nothing.

Example: `void printMessage(char *message) { printf("%s\n", message); }`

2

1. parameterized functions :-

functions that take arguments in input as parameters.

Example: int sum(int a, int b);

2. No-parameter functions :-

no parameters are required.

Example: void printfHello();

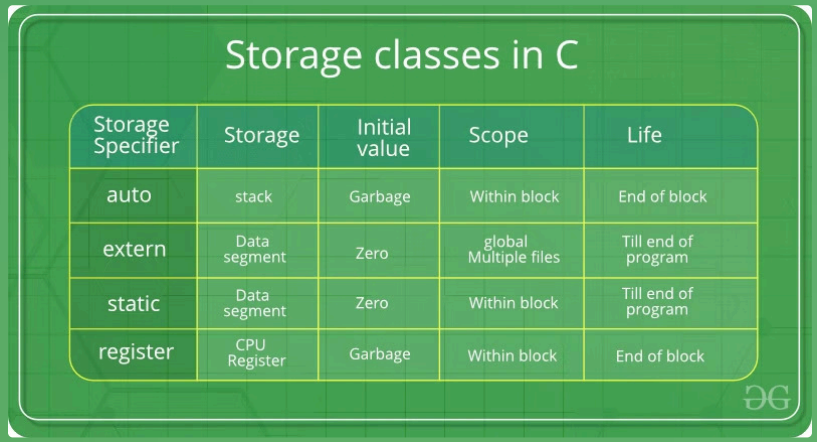
Scope and Lifetime of Variables

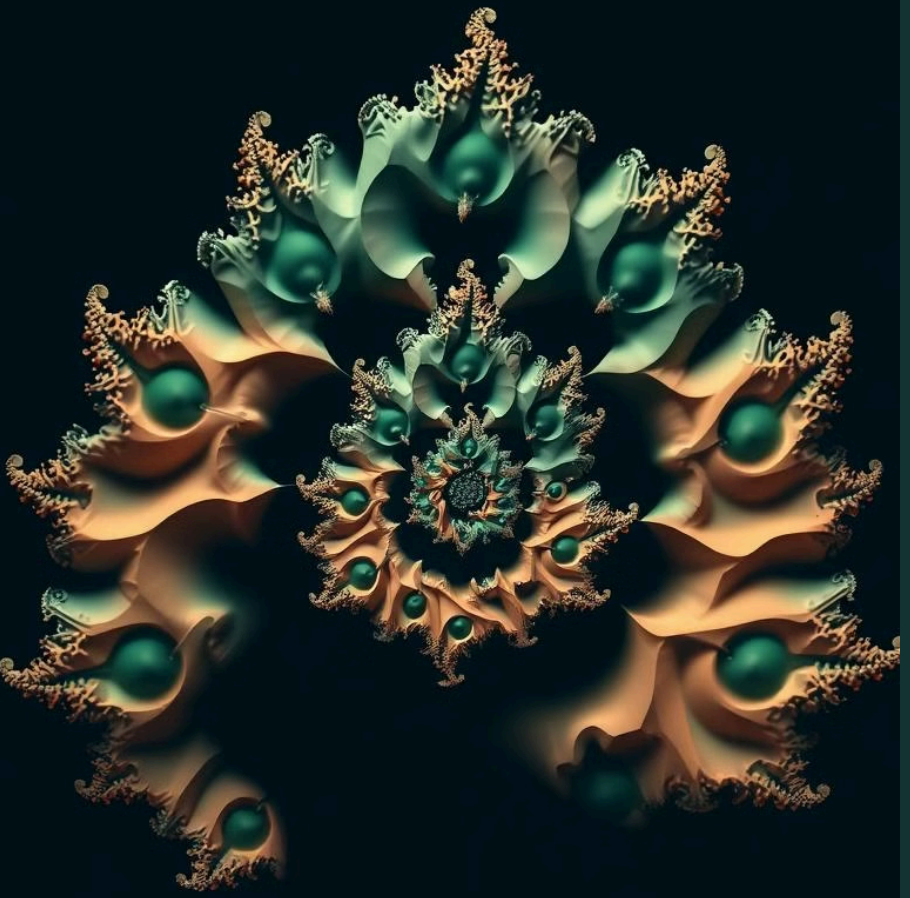
Local variables are inside functions. Global variables are accessible everywhere. Static variables retain value between calls.

Global

Local

Static





Introduction to Recursion

Recursion: a function calls itself. A base case stops the recursion. The recursive step modifies the input. Prevent infinite recursion.

1

Base Case

Stops recursion.

2

Recursive Step

Calls itself with modified input.

Recursive Factorial Calculation

Factorial: n! = n * (n-1) * ... * 1. The recursive function calls itself to calculate each step. The base case is 0! = 1.

```
int factorial(int n) {
  if (n == 0) {
    return 1; // Base case
  } else {
    return n * factorial(n - 1); // Recursive step
  }
}
```

```
dacocntfaral(‹ fabl 3)
rocedleraital = vit.1..)
    cate(db);
    pocetifall, reccbredight);
    redumaticall, att tatalinoboltialibil)

    peachi :
1   tnassift faxt. 1b:
2   tucesifal, relalbisinton;
3   tuassift in;
4   tucesift taxt.ib:
5   tuassift fart.if:
4   tucesifratall fartl ib:
7   recesifal fact.iral, creatbil,
7   tuceslialtingmnililt;
4   aleatricalay;?;:
5   rocefiral colltarl, ccaltenthly,
8   - seristionalt catiefurlitht..))
```