Lecture 4 (2nd part) Recursion

Dr. Hacer Yalım Keleş

Ref: Programming in ANSI C, Kumar

RECURSION

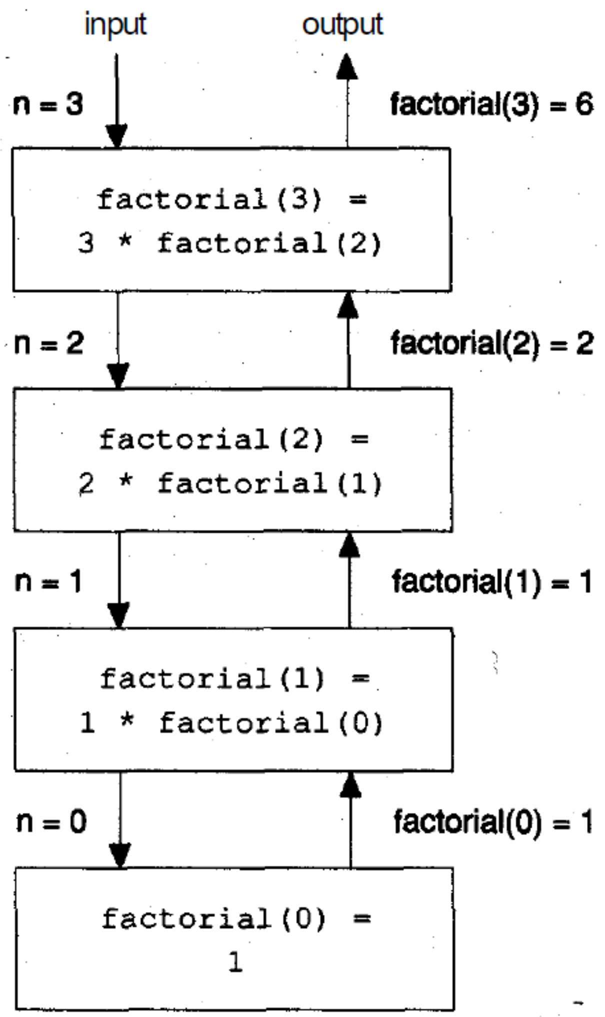
Recursion is the process whereby a construct operates on itself. In C, functions may be defined recursively; that is, a function may directly or indirectly call itself in the course of its execution. If the call to a function occurs inside the function itself, the recursion is said to be *direct*. However, if a function calls another function, which in turn makes a call to the first one, the recursion is said to be *indirect*.

The factorial function, defined as

factorial(O) = 1
factorial(n) = n* factorial(n-l), $n > 0$

is the classic example of recursion. This function can be coded as

```
int factorial(int n)
  {
    if (n == 0)
        return 1;         /* termination condition */
    else
        return n * factorial (n-1);/* recurse */
  }
```



Recursive evaluation of factorial (3)

As another example of recursion, consider the Fibonacci sequence of numbers

1,1,2,3,5,8,13,21,...

defined by

fib(O) = fib(l) = 1
fib(n) = fib (n-1) + fib(n-2), *n* > 1

Please, try to write this recursive function.

As another example of recursion, consider the Fibonacci sequence of numbers

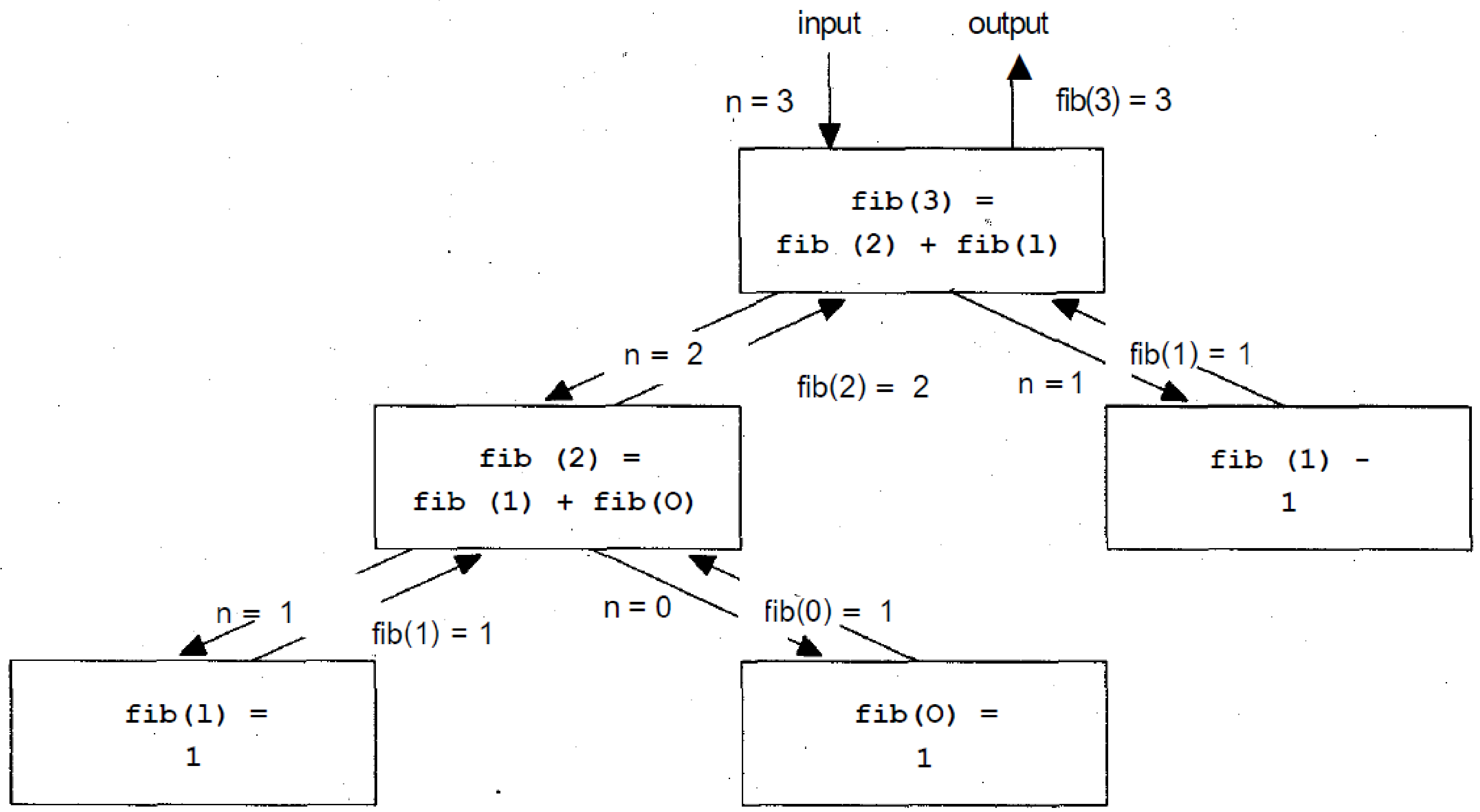
1,1,2,3,5,8,13,21,...

defined by

fib(O) = fib(l) = 1
fib(n) = fib (n-1) + fib(n-2), $n > 1$

This recursive definition can be translated into a recursive C function in a straightforward manner as

```
int fib(int n)
  {
    return n == 0 || n == 1 ? 1 : fib(n-l) + fib(n-2);

  }
```



Recursive evaluation of **`fib (3)`**

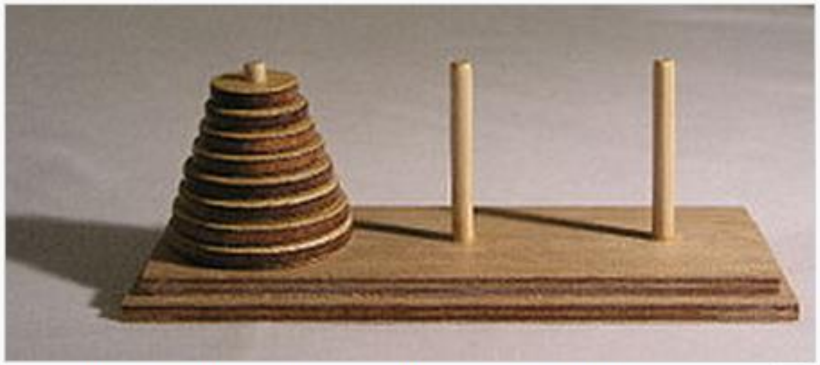
The following is a nonrecursive function for computing the nth Fibonacci number:

```
int fib(int n)
  {
    int i, result, n_minus1 = 1, n_minus2 = 1;

    if (n == 0 || n == 1) return 1;

    for (i = 2; i <= n; i++)
      {
        result = n_minus1 + n_minus2;
        n_minus2 = n_minus1;
        n_minus1 = result;
      }

    return result;
  }
```

It is obvious that the recursive versions of these functions are more concise and understandable than their iterative counterparts, and their correctness can easily be verified by appealing to the corresponding mathematical definitions. Why would then one ever want to use iterative versions of these functions? The problem with the recursive solutions is that, compared to their iterative versions, they can be far more inefficient. Iterative versions are generally faster as they avoid the overhead of passing arguments and returning values in a series of function calls and returns. The other source of inefficiency in recursive solutions is redundant computations. For example, in calculating `fib (3)`, fib (1) is calculated two times, and in calculating fib(4),fib(2) is calculated two times, fib (1) three times, and fib (0) two times. As n increases, this duplication of calculation increases exponentially, resulting in prohibitive execution time.

Reading Assignment

- Read and implement Section 5.9.2 (Tower of Hanoi Puzzle)



A model set of the Towers of Hanoi (with 8 disks)

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk may be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
3. No disk may be placed on top of a smaller disk.