Recursive and non-recursive programs

For a universal programming language (like Pascal) recursion is, in a sense, redundant: for any recursive program it is possible to write an equivalent program without recursion. Of course, this does not mean that recursion should be avoided, because it allows us to provide elegant solutions to otherwise complicated problems.

However, we want to show some methods that allow us to eliminate recursion in some cases and transform a recursive program into an equivalent non-recursive program.

What for? A pragmatic answer is that sometimes recursion is implemented in a non-efficient way and recursive programs may be significantly slower than equivalent non-recursive programs. Another problem is that some programming languages do not allow recursion at all. But the main reason is that elimination of recursion is sometimes very instructive. In section 8.1 we describe a technique that often allows us not only to eliminate recursion, but also get a faster program. Then in section 8.2 we consider a more general approach. Finally, in section 8.3 we show some examples not covered by these techniques.

8.1 Table of values (dynamic programming)

8.1.1. The following recursive procedure computes binomial coefficients. Write an equivalent program without recursion.

```pascal
function C(n,k: integer):integer;
| {n >= 0; 0 <= k <= n}
begin
| if (k = 0) or (k = n) then begin
| | C:=1;
| end else begin {0<k<n}
| | C:= C(n-1,k-1)+C(n-1,k)
| end;
end;
```
Remark. \( C(n, k) = \binom{n}{k} \) is the number of \( k \)-element subsets of an \( n \)-element set. The identity
\[
\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}
\]
is proved as follows: Fix some element \( x \) of the \( n \)-element set. Then all \( k \)-element subsets are divided into two categories: those that contain \( x \) and those that do not. The elements of the first type are in one-to-one correspondence with the \((k - 1)\)-element subsets of a \((n - 1)\)-element set (just discard \( x \)); the elements of the second type are \( k \)-element subsets of a \((n - 1)\)-element set.

The table of \( \binom{n}{k} \)-values
\[
\begin{array}{cccc}
1 & & & \\
1 & 1 & & \\
1 & 2 & 1 & \\
1 & 3 & 3 & 1 \\
1 & 4 & 6 & 4 & 1 \\
\ldots & \ldots & \ldots & \ldots & \\
\end{array}
\]
is called the Pascal triangle (the same Blaise Pascal who gave his name to a programming language). In this triangle, any element (except the 1s on the left and the right) is the sum of the two elements above it.

Solution. One may use the formula
\[
\binom{n}{k} = \frac{n!}{k!(n-k)!}
\]
We do not use it because we want to show more general methods to eliminate recursion. Our program fills the table of values \( C(n, k) = \binom{n}{k} \) for \( n = 0, 1, 2, \ldots \) until it reaches the element in question.

8.1.2. Compare the computation time for the recursive and non-recursive versions of the binomial coefficient algorithm, and similarly for the amount of memory used.

Solution. The table used in the non-recursive version occupies space of order \( n^2 \). We can reduce it to \( n \) if we recall that only one line of the Pascal triangle is needed to compute the next line. The time required is still \( n^2 \).

The recursive program requires much more time. Indeed, the call \( C(n, k) \) causes two calls of type \( C(n-1, \ldots) \), those two calls cause four calls of type \( C(n-2, \ldots) \), etc. Hence, the time is exponential (of order \( 2^n \)). The recursive procedure uses \( O(n) \) memory (we have to multiply the recursion depth, that is \( n \), by the amount of memory required by one copy of the procedure, that is \( O(1) \)).

The reason why the non-recursive version is so much faster is the following. In the recursive version, the same computations are repeated many times. For example, the call \( C(5, 3) \) causes two calls of \( C(3, 2) \):