# 13 THE PROBLEM OF THE NEXT PERMUTATION

We are requested to write an inner block operating on a global integer array variable, named $c$, with

$$c.lob = 1 \quad \text{and} \quad c.hib = n$$

for some constant value of $n$ ($> 1$). Furthermore it is given that the ordered sequence of values $c(1), c(2), \ldots, c(n)$ is some permutation of the values from 1 through $n$, but **not** the alphabetically last one: $n, n-1, \ldots, 1$. The inner block has to transform the sequence $c(1), c(2), \ldots, c(n)$ into its immediate alphabetic successor. (For the notion of "alphabetic order", see the last example of the chapter "The Formal Treatment of Some Small Examples".) For instance, with $n = 9$, the sequence

$$1\,4\,6\,2\,9\,5\,8\,7\,3$$

should be transformed into

$$1\,4\,6\,2\,9\,7\,3\,5\,8$$

As the above example shows, we may have at the low end a number of function values that remain unaffected. The transformation to be performed is restricted to permuting the values at the high end and our first duty seems to be to find that split, i.e. to determine the value of $i$, such that

$$c(k) \text{ remains unaffected for } 1 \leq k < i$$

$$c(k) \text{ is changed for } k = i$$

That value of $i$ is characterized as the maximum value of $i$ ($< n$) such that

$$c(i) < c(i+1)$$

(It could not be larger, for then we would be restricted to permuting an initially monotonically decreasing sequence, an operation that cannot give rise to a sequence that is higher in the alphabetic order; it should not be smaller either, because then we would never generate the immediate alphabetic successor.)

*Note.* The fact that the initial sequence is not the alphabetically last one guarantees the existence of an $i$ $(0 < i < n)$ such that $c(i) < c(i + 1)$. (*End of note.*)

Having found $i$, we must find from "the tail", i.e. among the values $c(j)$ with $i + 1 \leq j \leq n$, the new value $c(i)$. Because we are looking for the immediate successor, we must find that value of $j$ in the range $i + 1 \leq j \leq n$, such that $c(j)$ is the smallest value satisfying

$$c(j) > c(i)$$

Having found $j$, we can see to it that $c(i)$ gets adjusted to its final value by "$c$:*swap*$(i,j)$". This operation has the additional advantage that the total sequence remains a permutation of the numbers from 1 through $n$; the final operation is to rearrange the values in the tail in monotonically increasing order. The overall structure of the program we are considering is now

determine $i$;

determine $j$;

$c$:*swap*$(i,j)$;

sort the tail

(In our example $i = 6$, $j = 8$ and the final result would be reached via the intermediate sequence *1 4 6 2 9 7 8 5 3*.)

When determining $i$, we look for a maximum value of $i$; the Linear Search Theorem tells us that we should investigate the potential values for $i$ in decreasing order.

When determining $j$, we look for a minimum value $c(j)$; the Linear Search Theorem tells us that we must investigate $c(j)$ values in increasing order. Because the tail is a monotonically decreasing function (on account of the way in which $i$ was determined), this obligation boils down to inspecting $c(j)$ values in decreasing order of $j$.

The operation "$c$:*swap*$(i,j)$" does not destroy the monotonicity of the function values in the tail (prove this!) and "sort the tail" reduces to inverting the order. (In doing so, our program "borrows" the variables $i$ and $j$ that have done their job. Note that the way in which the tail is reflected works equally well with an even number as with an odd number of values in the tail.)

```
begin glovar c; privar i, j;
  i vir int:= c.hib − 1; do c(i) ≥ c(i + 1) → i:= i − 1 od;
  j vir int:= c.hib; do c(j) ≤ c(i) → j:= j − 1 od;
  c:swap(i,j);
  i:= i + 1; j:= c.hib;
  do i < j → c:swap(i,j); i,j := i + 1, j − 1 od
end
```

*Remark 1.* Nowhere have we used the fact that the values $c(1), c(2), \ldots, c(n)$ were all different from each other. As a result one would expect that this program would correctly transform the initial sequence into its immediate alphabetic successor also if some values occurred more than once in the sequence. It does indeed, thanks to the fact that, while determining $i$ and $j$, we have formed our guards by "mechanically" negating the required condition $c(i) < c(i+1)$ and $c(j) > c(i)$ respectively. I once showed this program, when visiting a university, to an audience that absolutely refused to accept my guards with equality included. They insisted on writing, when you knew that all values were different

$$\textbf{do } c(i) > c(i+1) \rightarrow \ldots$$

and

$$\textbf{do } c(j) < c(i) \rightarrow \ldots$$

Their unshakable argument was "that it was much more expensive to test for equality as well". I gave up, wondering by what kind of equipment on the campus they had been brainwashed. (*End of remark 1.*)

*Remark 2.* Programmers unaware of the Linear Search Theorem often code "determine $j$" erroneously in the following form:

$$j \textbf{ vir } int:= i + 1; \textbf{ do } c(j+1) > c(i) \rightarrow j:= j + 1 \textbf{ od}$$

They argue that this program will only assign to $j$ the value $j + 1$ after it has been established that this new value is acceptable in view of the goal $c(j) > c(i)$. Analyze why their version of "determine $j$" may fail to work properly. (*End of remark 2.*)

*Remark 3.* One time I had the unpleasant obligation to examine a student whose inventive powers I knew to be strictly limited. Because he had studied the above program I asked him to write a program transforming the initial permutation, known not to be the alphabetically first, into its immediate alphabetic predecessor. I hope that this exercise takes you considerably less than the hour he needed. (*End of remark 3.*)

*Remark 4.* This program is a particular friend of mine, because I remember having tackled this problem in my student days, in the Stone Age of

machine code programming (even without index registers: in the good old von Neumann tradition, programs had to modify their own instructions in store!). And I also remember that, after a vain struggle of more than two hours, I gave up! And that at a moment when I was already an experienced programmer! A few years ago, needing an example for lecturing purposes, I suddenly remembered that old problem and solved it without hesitation (and could even explain it the next morning to a fairly inexperienced audience within twenty minutes). That now one can explain within twenty minutes to an inexperienced audience what twenty years before an experienced programmer could not find shows the dramatic improvement of the state of the art (to the extent that it is now even hard to believe that then I could not solve this problem!). (*End of remark 4.*)

*Remark 5.* Equivalent to our criterion for $i$ ("the maximum value of $i$ ($< n$), such that $c(i) < c(i + 1)$") is "the maximum value of $i$ ($< n$) such that $(\mathbf{E}\, j: i < j \leq n: c(i) < c(j))$". The latter criterion is, however, less easily usable and whoever starts with the latter one had better discover the other one (in one way or another). (*End of remark 5.*)