P. Bratley - Permutation with Repetitions

Comments on:

Bratley, P. (1967). Algorithm 306: Permutation with Repetitions [G6]. *Communications of the ACM* **10**(7), July 1967, pp. 450-451.

Is the ordering of the labels (1-12) in that order for a reason?

Even though there were a few lines of code, it took me a while to read through the algorithm. The fact that L1 to L12 do not appear in ascending order made it a little bit harder for me to read. Is there any special reason why the L's are not organized in ascending order?

Why not in order?

You know, I expect that it's a design decision, but I'm not sure particularly why he made that design decision.

Is "own integer" somewhat equivalent to a static integer variable?

Yes. It's a variable that is local to the procedure, but that persists across multiple calls.

This algorithm is unbelievably confusing to interpret. Do you think it was just as confusing to write as it is to read?

No, I think the author had a clear sense as to how the algorithm would work, and expressed it that way. Of course, getting it perfectly right was clearly hard, which we know because it had to be rewritten and because it still does not meet its stated goal of putting the results in order.

What do the numbers after the L mean,e.g L12?

Those numbers are parts of the labels.

When they programed this, did they naturally come up with those many gotos? If not, why would they choose to use this many in the paper?

I think you've answered your own question - It's unlikely that someone would insert this many goto statements and labels unless it was the way they regularly programmined.

Algorithm 306 definitely showcases how complicated an algorithm can be with goto statements. It is quite difficult to decipher how the data flows through the algorithm. Aside from the trouble I had finding out what exactly this algorithm tries to accomplish (due to what I consider poor documentation), it is unclear to me how these go to statements work. For example, if I go to L12 and it's an assignment satement, should I just move to the next line? Is the command after a goto ever read? For example, in line 9 of the algorithm, one can go to L99 and then "end". Isn't that last "end" superflous since it might not be read and

L99 leads to the end of the algorithm anyway?

A goto statement simply transfers control to the given label. Any code after the label is executed. The end statement is necessary, because you have to mark the end of an algorithm. (Remember, the algorithm would appear with other code, so the end of one and the start of the next must be distinguished.)

I understand what the algorithm does and returns by means of trying different inputs, but in general thought it was very complicated and confusing. For example, L6 could very easily have been implemented with a for loop, similarly L4, L5. Why did the writer of this code, want to use so many gotos, could it not be implemented any other way then (1967)?

I do not know about the author's design decisions. However, if you look at the code more closely, you'll see that things that look like loops are a bit more subtle. For example, the L6 loop returns to the top in two different places places.

A shining example of Dijkstra's point in the previous article. After implementing this algorithm in C, I didn't know how it worked. After reimplementing it without the use of GOTO, I still didn't understand it. Only after I sat down and analyzed it with the intent of not looking like an idiot on monday did I finally grasp what it is trying to do. While the pseudocode is concise, the use of GOTO statements obfuscates the underlying algorithm to the point where it is barely above gibberish.

You may be the only one in the class who now knows what the algorithm does.