15 June 1967

A sequel to EWD200.

In today's letter I shall deal with a subtle question, viz. the problem of identity of elements in a set, a population, of variable size. Mind you, that in the opening paragraph of EWD200 I have spoken of components "with a permanent existence, an undeniable identity", and indeed, if we have a set of 100 elements, numbered (or should I say "identified"?) from 0 through 99, we have no problem.

But suppose that this is a vector that is allowed to shrink and grow in stack fashion and let element nr.99 be removed from the top of the stack. This suggests, that the remaining elements (with a permanent existence, an undeniable identity!) keep their identity index, i.e. remain identified by their distance from the stack bottom. This is essentially what is meant by "identity", it provides a means in which we can refer (and continue to refer) to the remaining elements.

But if as a next move, a new element is added to the stack, we can raise the question, what the identity index of the element just added will be. There are at least two different attitudes:
a) in the stack elements are identified by their distance from the bottom, therefore the element just added will get as identity index the value 99
b) the element just added is a new object, definitely different from the old top element, that has ceased to exist; because somewhere we might (probably erroneously) still have hanging around a now obsoleet reference to the no longer existing old top, it is essential the the new top will get a different identity, i.e. an identity index that is still "virgin" with respect to this population (say: 100).
(There is even a solution in between: if it is possible to scan the indices that might refer to elements in the population, one can upon removal of element nr 99 scan for this value and replace it, whenever found, by some value meaning "undefined" so that the new top can again get the identity index 99 without the danger of identifying with each other the old and the new top.)

The whole point in this discussion is to make clear that population shrinking or growing are undefined operations, unless we define the algorithm that controls the generation of new identities, of new identity indices when the thing grows. And we have given two examples of such algorithms, case a):the lowest free index value and case b): the lowest virgin index value.

The first conclusion is that if we allow our variable size machine to vary in some aspect of its size, this is only a well defined operation provided we have decided upon the pattern according to which identity indices for the elements to be added will be created. This decision is not trivial.

In designing our variable size machine we must make such a choice for the standard patterns according to which it may vary. My proposal is to keep method b) out of the hardware, for I shudder at the thought. In the case of a stack I prefer method a) for a large variety of reasons.

For one thing: method a) is much easier to implement. Furthermore method a) gives a more practical significance to the identity index, because the elements "under" or "on top of" an identified element have identity indices that can be generated by subtraction and addition. Finally, if we describe stack arithmetic, we can describe in words the binary operation on the top in two different ways, say the sum. We can say "The top two elements are removed from the stack and the sum of the values removed is then put on the top of the stack", a description in which the stack elements seem to be constant during their life time, or we can say "The top element is removed from the stack and the then current top is replaced by the sum of its original value and that of the value removed". Method a) has the advantage that both descriptions of the addition give the same identity to the stack element whose value equals the sum.

The final remark is that method b) ignores the fact that the varying size object is indeed a stack!

In my present stage of thinking I restrict myself mentally to size variations in which the identity indices with respect to a certain size aspect (such as a vector) will have consecutive values. For the sake of completeness I mention two consequences for the program to be carried out by the variable size machine.

The first consequence is that in the case of wilder life times we may have inside the variable size machine unused portions. If the aspect in question is storage and we think the large size too expensive (this, I am sorry to admit, is a hard question to decide!) then the program should have the decency to do some garbage collection, shoul move information (physically, I presume) and should reduce XXXXXXXX the size of its machine.

A next consequence is, that the program inside the variable size machine has the responsibility to see, that obsoleet references to no longer existing objects do not lead the process astray. Our decision is, that also this will not belong to the responsibility of the system implementing the variable size machine.

In order not to confuse the issue, we continue to think today about machines of which only one aspect of the size can vary, and in order to be explicit, we consider again a vector of storage locations.

The choice of method a) implies then that we can give another interpretation of the variable aspect: we can regard the situation as if the program were working all by itself in a very large (may be "infinite" in one of the appropriate meanings) memory, all the time stating as well lower and upper bound of the area used. It is another picture, may be it will provide in the long run an easier terminology. Up till now I don't think so.

There is a very hard efficiency problem, that I shall indicate now, without looking for an answer. One may expect that the introduction of a constant size aspect can be done much cheaper then the introduction of a variable size aspect. If so, it might be cheaper (if a certain aspect does not vary too much and an a priori upper bound of the size is known) to introduce immediately an constant size aspect, leaving parts of the machine unused. This question would be solved if we find that a variable size aspect can be implemented so satisfactorily, that it does not pay to introduce the fixed size aspect as a special class.

There is a second efficiency problem. Undoubtedly it will be expensive to have the variable size machine too large for a long period of time; on the other hand the processing of the redefinitions of the size might be expensive. In the case of a stack this can be overcome: there is no need to let the machine size follow all microscopic changes in stack length, one can decide that machine size variations will XX take place in larger grains. The trouble is the size of this grain and (if the most appropriate grain size is a system characteristic) the variable size machine can be programmed independent of the actual grain size. Reference to an object that is not contained in the machine is clearly nonsense and will be detected; that reference to an object inside the machine can be meaningless (due to "majoration") is a possibility. If so, this imposes a responsibility on the particular program, not unlike the previous ones.