ON THE COMPACTIFICATION OF PROGRAMS

K.R.Apt, C.Bongers and M. Stefanski
Faculty of Economics
Erasmus University
P.O.Box 1738, 3000DR Rotterdam
The Netherlands

It has been argued in the literature that a natural and effective way to obtain programs of good quality is to proceed through the process of stepwise refinement. We show in this short note how this technique can be applied to obtain compact and succinct programs devoid of superfluous assignments or multiple loops. To illustrate our point we chose the well known Josephus problem taken from Knuth [1]:

There are n men arranged in a circle. Beginning at a particular position, we count around the circle and brutally execute every third man (the circle closing as men are decapitated). Try to design a clever algorithm to determine which two are left (it may save your life).

While trying to solve this problem we should at first settle on the choice of the programming language in which the solution should be coded. Since our goal is to obtain a short and simple solution we have to exclude FORTRAN or PASCAL because of their lack of provision for dynamic arrays which, as we shall soon see, are needed here. On the other hand languages like Algol 60, Algol 68, PL/I and Ada are unsatisfactory because of their requirement of declaring all variables which in addition to all those BEGINs and ENDs and identifiers of more than one letter unnecessarily lengthens the program text. As a result we are naturally led to choosing BASIC.

Our idea is to use an array L of N (N>=3) elements to form a ring of N elements from which every third element is removed until two elements are left over. This brings us to the following 12 line program

```
0010 INPUT N
0020 DIM L[N]
0030 FOR K=1 TO N-1 STEP 1
0040 LET L[K]=K+1
0050 NEXT K
0060 L[N]=1
0070 K=1
0080 FOR I=1 TO N-2 STEP 1
0090 LET L[L[K]]=L[L[L[K]]]
0100 LET K=L[L[K]]
0110 NEXT I
0120 PRINT K,L[K]
```

The first loop together with line 60 is used here to form the ring whereas the second is used to leave out every third element.

An astute programmer will immediately observe various natural improvements which lead to the following 9 line version

```
0010 INPUT N
0020 DIM L[N]
0030 FOR K=N TO 1 STEP -1
0040 LET L[K]= K+1-INT(K/N)*N
0050 NEXT K
0060 LET L[L[K]]=L[L[L[K]]]
0070 LET K=L[L[K]]
0080 IF K<>L[L[K]] THEN GOTO 0060
0090 PRINT K,L[K]
```

Note the elegant use of the fact that the variable K equals 1 in most versions of BASIC upon exit from the first loop. This technique used here to save one assignment cannot be employed in PASCAL where upon exit from the loop the value of the loop index is undefined. Furthermore, the program demonstrates once again the power of GOTO.

A closer look at the last program still reveals some deficiencies. Even though we saved here one line by using the GOTO statement instead of the for loop we are still left with two loops in the program. The major question now is: is it possible to merge these two loops into one, thereby further reducing the length of the program, and eliminating redundancy?

While looking for an answer to this question it is illuminating to observe that both loops consist of updating the array L and computing the new value of K (in that order). The next version of the program makes use of this fact and also employs the useful feature of BASIC that all variables are initialized to zero.

In the above program the values assigned to K are: N (at the beginning), K-1 (during the first loop) and L[L[K]](during the second loop). On the other hand the subscript of L used during the first loop is K and during the second loop L[K] and the values assigned are K+1-INT(K/N)*N and L[L[L[K]]], respectively.

This suggests that the two assignments to K and L should be of the following form:

```
K=A*(N+1)+B*(K-1)+C*L[L[K]]
L[B*K+C*L[K]]=B*(K+1-INT(K/N)*N) +C*L[L[L[K]]]
```

where

A is 1 at the first execution of the K-assignment, thereafter it equals 0,

B is 1 throughout the first loop and 0 throughout the second loop,

C is 0 throughout the first loop and 1 throughout the second loop.

A proper choice of the values is:

```
A=1-SGN(L[N]),
B=1-SGN(L[0]),
C=SGN(L[0]).
```

The program would now be of the following form:

```
0010 INPUT N
0020 DIM L[N]
0030 assignment to K
0040 assignment to L
0050 IF K<>L[L[K]] THEN GOTO 30
0060 PRINT K,L[K]
```

Let us analyze what happens during the execution of this program. During the first phase the corresponding assignments to L are carried out in the following order: L[N]=1, L[N-1]=N,..., L[1]=2 and finally L[0]=1. Now the second phase begins. C now equals 1 and K becomes L[L[0]] i.e. 2 and not 1 as we would wish. (The reader is asked to consult the second version of the program). Thus we start the second phase with a wrong value of K. This difficulty can be solved by assigning to L the value B*(K+1-INT(K/N)*N) +C*L[L[L[K]]]+D
where D is N-1 if K is 0 and 0 otherwise. Thus a choice D=(1-SGN(K))*(N-1) will do.

Now L[0] becomes N, so L[L[0]]=1 as desired. The second phase begins with the assignment K=1 . Note that the test at line 70 is true only at the very end of the second phase when there are only two elements left in the ring. These two elements are the desired ones.

Summarizing, the complete listing of the program looks as follows:

```
0010 INPUT N
0020 DIM L[N]
0030 LET K=(1-SGN(L[N]))*(N+1)+(1-SGN(L[0]))*(K-1)+
     SGN(L[0])*L[L[K]]
0040 LET L[(1-SGN(L[0]))*K+SGN(L[0])*L[K]]=(1-SGN(L[0]))*
     (K+1-INT(K/N)*N)+SGN(L[0])*L[L[L[K]]]+(1-SGN(K))*(N-1)
0050 IF K<>L[L[K]] THEN GOTO 30
0060 PRINT K,L[K]
```

REMARK:One might object to the use of the fact all variables are initialized to zero. This problem can be avoided by observing that the desired initial assignment can be performed by the user provided he is encouraged to do so. For example the initial assignment of K to N might be achieved by replacing the first line of the last program by

```
INPUT "TYPE IN THE NUMBER OF PRISONERS (TWICE TO AVOID
ERRORS)";K,N.
```

An appropriate refinement of the program which uses this technique to assign initial values to both K and L[0] is left to the reader.

The final version of the program has only 6 lines and is several orders of magnitude simpler and more transparent than the first one. Any improvement in this matter can be achieved only by some additional insights into the problem. We encourage the reader to find a solution to the problem which has only 5 lines. We do have such a solution and shall present it in the next issue of this Bulletin.

CONCLUSIONS : We hope to have demonstrated that BASIC possesses various, largely unexplored APL-like features which can be nicely combined with the process of stepwise compactification. At the same time we have introduced a new powerful technique of loop merging which deserves a further study.

REFERENCE

[1] Knuth, D., The Art of Computer Programming, Vol. 1
Addison-Wesley, 1973