This note has appeared as THT-TW-Memorandum 249, March 1979,

by Maarten M Fokkinga

This note is a corrected version of TW-Memo 249.

A SIMPLER CORRECTNESS PROOF OF AN IN-PLACE PERMUTATION ALGORITHM.

Abstract. In 1972 Duijvestijn gave a correctness proof of a particular permutation algorithm using an invariant relation. We present another proof based on this relation. It uses ghost variables and consequently can be split up into easily comprehensible parts. This might be of interest to the reader. The verification itself is hardly of interest: the application of the predicate transformation rules is straightforward and involves nearly no mathematics.

The program. The program to be proved correct rearranges a variable v: array [0 .. n-1] of T with initial value V , according to a given permutation F of 0 .. n-1 , using only auxiliary variables which do not depend on n or on T . Thus the program establishes

R: A i: 0 .. n-1 . v(i) = V(F(i)) .

Using here and in the sequel the notation of (Dijkstra 76), the program reads

```
j := 0;
do j ≠ n-1 →
   q:=F(j); do q<j → q:=F(q) od;
   v :swap(j,q); j:=j+1
od .
```

We will give the correctness proof together with a construction of the program. We want to stress again that the verfication of the invariant relations is merely a boring formula manipulation, involving no interesting mathematics. We present it only to contrast it with (Duijvestijn 72).

A "stepping stone" program: using a ghost variable

We try to establish R by means of a repetition with the invariant relation (found by standard techniques)

$0 \le j < n$ and A i: 0 .. j-1. v(i) = V(F(i)) .

In order to know how the remaining elements of v need to be arranged yet, we introduce an array variable f , representing a permutation of j .. n-1 , such that

A i: j .. n-1. v(f(i)) = V(F(i)) .

Letting f(i) = i for i: 0 .. j-1 , we can express the complete invariant relation as P0 and P1 :

P0: f denotes a permutation of $0 .. n-1$,

P1: $0 \le j < n$ <u>and</u> <u>A</u> i: $0 .. n-1.\ v(f(i)) = V(F(i))$ <u>and</u> <u>A</u> i: $0 .. j-1.\ f(i) = i$.

The program, then, has the structure

  f := F; j := 0 {v = V; hence P0 <u>and</u> P1 established};

  <u>do</u> "maintain P0 <u>and</u> P1, decrease n-j" <u>od</u> {R} .

There are several ways to derive or invent the refinement "maintain P0 <u>and</u> P1, decrease n-j":

  $j \ne n-1 \rightarrow v$:swap(j,f(j));f :swap(f^{-1}(j),j); j := j+1 .

We will give the verification of the invariance only.

Recall the semantics of assignment and swapping.

$wp(x := e, P) = P[x \leftarrow e]$.

$wp(a :swap(x,y), P) = P[a \leftarrow a']$, where $a' = a[x \leftarrow a(y), y \leftarrow a(x)]$.

In general, the array value $a' = a[x \leftarrow e1, y \leftarrow e2]$ is defined for any a, x, y, e1, e2 with $x \ne y$ <u>or</u> $e1=e2$, as follows

$$a'(i) = \begin{cases} a(i) & \text{for i different from x and y} \\ e1 & \text{for } i = x \\ e2 & \text{for } i = y \text{ .} \end{cases}$$

Now we prove the invariance; first of P0 and then of P1.

Because f is subject to swap only, P0 is kept invariant. Formally this is shown as follows.

$wp(v :swap(j, f(j));\ f :swap(f^{-1}(j),j);\ j := j+1,\ P0) =$

$= ((P0[j \leftarrow j+1])[f \leftarrow f'])[v \leftarrow v']$

  where $f' = f[f^{-1}(j) \leftarrow f(j),\ j \leftarrow f(f^{-1}(j))]$, $v' = ...$

= f' is a permutation of $0 .. n-1$

= $f \circ p$ is a permuation, where p is the pair exchange "$f^{-1}(j)$ <--> j"

and this holds true, because f being a permutation on account of P0, and p being a permutation, so is the composition $f \circ p$.

(Note that, f being a permutation, the inverse f^{-1} is well defined!)

$wp(v :swap(j, f(j));\ f':swap(f^{-1}(j), j);\ j := j+1,\ P1) =$

$= ((P1[j \leftarrow j+1])[f \leftarrow f'])[v \leftarrow v']$

  where $f' = f[f^{-1}(j) \leftarrow f(j),\ j \leftarrow f(f^{-1}(j))]$

    $v' = v\ [j \leftarrow v(f(j)),\ f(j) \leftarrow v(j)]$

$= 0 \le j+1 < n$ <u>and</u> <u>A</u> i: $0 .. n-1.\ v'(f'(i)) = V(F(i))$ <u>and</u> <u>A</u> i: $0 .. j.f'(i) = i.$

The first term is implied by P1 and $j \neq n-1$; we prove

a: $v'(f'(i)) = V(F(i))$, and

b: $i>j$ or $f'(i) = i$

from P1 by cases on i:

For $f^{-1}(j) \neq i \neq j$:

a. $v'(f'(i)) =$ (def f':) $v'(f(i)) =$ (def v':) $v(f(i)) =$ (from P1:) $V(F(i))$,

b. $f'(i) =$ (def f':) $f(i) =$ (from P1:) i, if $i \leq j$.

For $i = j$:

a. $v'(f'(j)) =$ (def f':) $v'(j) =$ (def v':) $v(f(j)) =$ (from P1:) $V(F(j))$,

b. $f'(j) =$ (def f') j .

For $i = f^{-1}(j)$:

a. $v'(f'(i) =$ (def f':) $v'(f(j)) =$ (def v':) $v(j) = v(f(f^{-1}(j))) =$ (from P1:) $V(F(i))$,

b. (from P1:) A i: 0 .. j-1. $f(j) = i$, hence $i = f^{-1}(j) \geq j$. Now either $i = f^{-1}(j) > j$, or $i = f^{-1}(j) = j$ and $f'(i) = f'(j) = j = i$.

Final program: the ghost variable eliminated

There is an additional invariant relation, which enables us to eliminate variable f :

P2: A i: j .. n-1. $f(i) =$ first elt in the seq $F(i), F^2(i), F^3(i)$ ... which is $\geq j$.

Here follows the proof of the invariance of P2.

wp(v :swap(j, f(j)); f :swap($f^{-1}(j)$, j); j := j+1, P2) =

= ((P2[j ← j+1])[f ← f'])[v ← v']

where $f' = f[f^{-1}(j) \leftarrow f(j), j \leftarrow f(f^{-1}(j))]$

and $v' = v[j \leftarrow v(f(j)), f(j) \leftarrow v(j)]$

= A i: j+1 .. n-1. $f'(i) =$ first elt in the seq $F(i), F^2(i), F^3(i)$ ... which is $\geq j+1$.

We prove the requirement for $f'(i)$ from P2 by cases on i.

For $j+1 \leq i \leq n-1$ and $i \neq f^{-1}(j)$:

$f'(i) =$ (def f':) $f(i)$ {and this is > j on account of P0 and P1}

= (from P2) the first elt in $F(i), F^2(i)$ ... which is $\geq j$,

so $f'(i) =$ the first elt in the seq $F(i), F^2(i)$ ... which is $\geq j+1$.

For $j+1 \leq i \leq n-1$ and $i = f^{-1}(j)$:

$f'(i)$ = (def f':) $f(j)$ {and this is $> j$ on account of P0, P1 and $f^{-1}(j) \neq j$}
= (from P2:) the first elt in $F(j)$, $F^2(j)$ ... which is $\geq j$,
so $f'(i)$ = the first elt in the seq $F(j)$, $F^2(j)$ ... which is $\geq j+1$ ... (*)
Also $j = f(f^{-1}(j)) = f(i)$
= (from P2:) the first elt in $F(i)$, $F^2(i)$ ... which is $\geq j$ ... (**)
Combining (*) and (**) yields
$f'(i)$ = the first elt in the seq $F(i)$, $F^2(i)$ ... which is $\geq j+1$.
This, by the way, is the most non-trivial step of all verifications.

Hence, just before v :swap(j, f(j)) we may compute f(j) as follows:
q:= F(j); <u>do</u> q<j → q:=F(q) <u>od</u> {q=f(j)} .
The invariant relation of the repetition reads:
$f(j)$ = the first elt in the seq q, $F(q)$, $F^2(q)$ ... which is $\geq j$.
The verification is easy, and is left to the reader. In addition P0, P1, P2, and $j \neq n$ are invariant as well, because they do not contain q .

Once the above line has been inserted, and v :swap(j, f(j)) has been replaced by v :swap(j, q) , it appears that f is not used at all -- except in updatings of itself -- and may therefore be deleted. So we have proved the correctness of the given program.

<u>In conclusion</u>. The ghost variable f has enabled us to split the program construction and the invariant relation in two easily comprehensible and separately verifiable parts. The preliminary mathematical properties proved by (Duijvestijn 72) have, more or less, been verfied during the straightforward and, indeed, rather boring verification of the invariants. Thus the only interesting feature of the correctness proof is the formulation of an elegant invariant.

<u>References</u>

Dijkstra, E.W.: A Discipline of Programming, Prentice Hall (1976).
Duijvestijn, A.J.W.: Correctness proof of an in-place permutation, BIT <u>12</u> (1972) 318-324.

BIT 12 (1972), 318–324

CORRECTNESS PROOF OF AN IN-PLACE PERMUTATION

A. J. W. DUIJVESTIJN

Abstract.

The correctness of an in-place permutation algorithm is proved. The algorithm exchanges elements belonging to a permutation cycle. A suitable assertion is constructed from which the correctness can be deduced after completion of the algorithm.

An in-place rectangular matrix transposition algorithm is given as an example.

Key words and phrases: Proof of programs, algorithm, program correctness, theory of programming.

Introduction.

The in-place permutation problem deals with the rearrangement of the elements of a given vector $VEC[i]$, $i=1(1)G$, $G \geqq 1$, using an arbitrary permutation $f(i)$ of the integers $1,\ldots,G$.

The problem that has to be solved is: write an algorithm that permutes the elements of VEC without using extra storage. That means if $VEC[i]=\alpha_i$ before the permutation then $VEC[i]=\alpha_{f(i)}$ after the permutation.

The solution of the permutation problem is given by the following algorithm:

```
procedure permute (VEC, f, G); value G; integer G; array VEC;
  integer procedure f;
comment f(x) is the index of VEC where the element can be found that has
  to be moved to VEC[x];
begin integer k, ko, kn, wr;
  for k := 1 step 1 until G do
  begin
    kn := f(k);
    for ko := kn while kn < k do kn := f(ko);
    if kn ≠ k then begin comment exchange (VEC[kn], VEC[k]);
```

Received Feb. 10, 1972. Revised April 11, 1972.


```
                  wr := VEC[kn]; VEC[kn] := VEC[k];
                  VEC[k] := wr
               end
  end
end
```

A special case of the permutation problem arises in the transposition of a rectangular matrix without using extra storage [2, 3]. In case the matrix $A[i,j]$, $i=1(1)m$ and $j=1(1)n$ is columnwise mapped onto a vector $VEC[k]$, $k=1(1)m*n$, $G=m*n$, the function f is defined as follows in *ALGOL*-60:

```
integer procedure f(x); value x; integer x;
comment f(x) is the index of VEC where the element can be found that has
  to be moved to VEC[x];
begin integer w;
  w := (x-1) ÷ n;
  f := (x-w*n-1)*m+w+1
end
```

The algorithm for which a correctness proof is given in this note is essentially that of R. F. Windley [1].

Correctness of the algorithm.

It has to be proved that the algorithm performs the following:

$$\forall i(1 \leqq i \leqq G \rightarrow VEC[i] = \alpha_{f(i)}) . \tag{1}$$

First we introduce a function $\varphi_k(i)$ that is defined for $k \leqq i \leqq G$ with $1 \leqq k \leqq G$:

$$\varphi_k(i) = \text{the first } f^{(s)}(i) \text{ with } f^{(s)}(i) \geqq k,\ s \geqq 1 .$$

The expression f^s means: f if $s=1$, otherwise $ff^{(s-1)}$. Consequently $\varphi_k(i) = f^{(s)}(i) \geqq k$, and $f^{(t)}(i) < k$ with $1 \leqq t < s$, $s \geqq 1$.

We prove certain properties of the function φ.

PROPERTY 1 is a property of the permutation f:

$$\forall i\bigl(1 \leqq i \leqq G \rightarrow \exists e1(1 \leqq e1 \leqq G \wedge i = f(e1))\bigr) .$$

and

$$\forall i\bigl(1 \leqq i \leqq G \rightarrow \exists e2(1 \leqq e2 \leqq G \wedge e2 = f(i))\bigr) .$$

PROPERTY 2.

$$(2)\qquad \forall i\big(k \leqq i \leqq G \rightarrow \exists e1(k \leqq e1 \leqq G \wedge i = \psi_k(e1))\big)$$

and

$$(3)\qquad \forall i\big(k \leqq i \leqq G \rightarrow \exists e2(k \leqq e2 \leqq G \wedge e2 = \psi_k(i))\big)\,.$$

PROOF. Let $V_{k,G}$ be the set of integers: $V_{k,G} = \{i: k \leqq i \leqq G\}$, then property 2 says that $\psi_k(i)$ is a permutation on $V_{k,G}$.

Apparently property 2 is true for $k=1$ since $\psi_1(i) = f(i)$ (property 1).

Assuming property 2 is true for k (induction assumption), we prove that property 2 is also true for $k+1$.

According to the induction assumption there exists exactly one element $e1 \in V_{k,G}$ such that $k = \psi_k(e1)$ and exactly one element $e2 \in V_{k,G}$ such that $e2 = \psi_k(k)$. (A direct consequence of (2) and (3)).

We consider two cases:

CASE 1. $e1 > k$. Then clearly $e2 > k$. Consider the sets $V^{*}_{k+1,G} = V_{k+1,G} \setminus e1$ and $V^{**}_{k+1,G} = V_{k+1,G} \setminus e2$.

According to the induction assumption we have:

$$(4)\qquad \forall a\big(a \in V^{*}_{k+1,G} \rightarrow \exists b(b \in V^{**}_{k+1,G} \wedge b = \psi_k(a))\big)$$

and

$$(5)\qquad \forall b\big(b \in V^{**}_{k+1,G} \rightarrow \exists a(a \in V^{*}_{k+1,G} \wedge b = \psi_k(a))\big)$$

Since $b = \psi_k(a) > k$ it follows from the definition of ψ:

$$b = f^{(s)}(a),\ s \geqq 1 \text{ and } f^{t}(a) < k \text{ for } 1 \leqq t < s$$

that

$$b = f^{s}(a) \geqq k+1,\ s \geqq 1 \text{ and } f^{(t)}(a) < k < k+1 \text{ for } 1 \leqq t < s;$$

$$(6)\qquad \text{we conclude } b = \psi_{k+1}(a)\,.$$

Hence it follows that:

$$(7)\qquad \forall a\big(a \in V^{*}_{k+1,G} \rightarrow \psi_k(a) = \psi_{k+1}(a)\big)\,.$$

Furthermore we prove $e2 = \psi_{k+1}(e1)$.

From the definition of ψ and the induction assumption it follows:

$$\exists s\big(s \geqq 1 \wedge k = f^{s}(e1) \wedge \forall t(1 \leqq t < s \rightarrow f^{t}(e1) < k)\big)$$

and

$$\exists r\big(r \geqq 1 \wedge e2 = f^{r}(k) \wedge \forall u(1 \leqq u < r \rightarrow f^{u}(k) < k)\big)\,.$$

Clearly $e2 = f^{s+r}(e1) \geqq k+1$, $s+r \geqq 2$ and $f^{p}(e1) < k+1$ with $1 \leqq p < s+r$. Hence

$$(8)\qquad e2 = \psi_{k+1}(e1)\,.$$

Using (4), (5), (6), (7) and (8) we conclude:

$$(9)\qquad \forall a\big(a \in V_{k+1,G} \rightarrow \exists b(b \in V_{k+1,G} \wedge b = \psi_{k+1}(a))\big)$$

and

$$(10)\qquad \forall b\big(b \in V_{k+1,G} \rightarrow \exists a(a \in V_{k+1,G} \wedge b = \psi_{k+1}(a))\big)\,.$$

CASE 2. $e1 = k$. In this case $e1 = e2 = k$. Furthermore $V^{*}_{k+1,G} = V^{**}_{k+1,G} = V_{k+1,G}$ and according to (4), (5), (6) and (7) we have:

$$(11)\qquad \forall a\big(a \in V_{k+1,G} \rightarrow \exists b(b \in V_{k+1,G} \wedge b = \psi_{k+1}(a))\big)$$

and

$$(12)\qquad \forall b\big(b \in V_{k+1,G} \rightarrow \exists a(a \in V_{k+1,G} \wedge b = \psi_{k+1}(a))\big)\,.$$

Using (9), (10), (11) and (12) then by induction property 2 is true for all $k \leqq G$.

We can now formulate property 3 and 4.

PROPERTY 3. If $\psi_k(e1) = k$ and $\psi_k(k) = e2$, while $e1 > k$ and $e2 > k$ then according to (8) $e2 = \psi_{k+1}(e1)$.

REMARK. In case $e1 = e2 = k$, $\psi_{k+1}(e1)$ is not defined.

PROPERTY 4. $\psi_k(i) = \psi_{k+1}(i)$ for all $i > k$ except that i for which $\psi_k(i) = k$ (see (6) and (7)).

We prove the truth of the assertion $E1 \wedge E2$ on a certain label in the program. The definition of $E1$ and $E2$ is as follows:

$$(E1)\qquad \forall i(1 \leqq i < k \rightarrow VEC[i] = \alpha_{f(i)})$$

and

$$(E2)\qquad \forall i(k \leqq i \leqq G \rightarrow VEC[\psi_k(i)] = \alpha_{f(i)})\,.$$

The structure of the program is:

```
for k := 1 step 1 until G do
begin ... end;
```

This program is equivalent with the program:

```
    k := 1;
L: if k > G then goto Exh;
    begin ... end;
    k := k+1; goto L; Exh:
```

We prove $\vdash E1 \wedge E2$ on label L for all k, $1 \leqq k \leqq G+1$.

PROOF. If $k=1$ then $\vdash E1 \wedge E2$ since $E1$ is true ($1 \leqq i < 1$ is false so the implication is true) and since $\psi_1(i)=f(i)$ the assertion $E2$ reads:

$$\forall i(1 \leqq i \leqq G \rightarrow VEC[\psi_1(i)] = VEC[f(i)] = \alpha_{f(i)}) \text{ which is clearly true .}$$

Assuming that $\vdash E1 \wedge E2$ on L for a certain $k=k_1$ ($1 \leqq k_1 \leqq G$) the following statements are executed before returning to label L.

```
L:  kn = f(k);
      for ko := kn while kn < k do kn := f(ko);
L1: if kn ≠ k then exchange (VEC[kn], VEC[k]);
L2: k := k+1; goto L;
```

The labels $L1$ and $L2$ are merely introduced as a reference. At label $L1$ we have $kn=\psi_k(k)$. Consequently $kn \geqq k$. In case $kn \neq k$, $VEC[kn]$ and $VEC[k]$ are exchanged. Since $\vdash E1 \wedge E2$ on L it follows $\vdash E1 \wedge E2$ on $L1$. We consider two cases:

CASE 1. $kn > k$. From $\vdash E1 \wedge E2$ on $L1$ we have $VEC[\psi_k(k)] = VEC[kn] = \alpha_{f(k)}$. After exchanging $VEC[kn]$ and $VEC[k]$, $VEC[k]=\alpha_{f(k)}$ at label $L2$.

Therefore the following assertion holds at $L2$:

$$\forall i(1 \leqq i \leqq k \rightarrow VEC[i] = \alpha_{f(i)}) .$$

Hence

$$\forall i(1 \leqq i < k+1 \rightarrow VEC[i] = \alpha_{f(i)}) .$$

Finally $\vdash E1$ at L for $k=k1+1$.

Since $kn=\psi_k(k) > k$ then according to property 2 there exist elements $e1$ and $e2$, $e1 > k$, $e2 > k$ such that:

$$e2 = \psi_k(k) \quad \text{and} \quad k = \psi_k(e1)$$

and according to property 3:

$$e2 = \psi_{k+1}(e1) .$$

Apparently $e2=kn$.

At label $L1$ we have

$$VEC[k] = VEC[\psi_k(e1)] = \alpha_{f(e1)}, \text{ since } e1 > k .$$

At label $L2$

$$VEC[kn] = VEC[\psi_k(k)] = VEC[e2] = \alpha_{f(e1)} .$$

Using property 3 at $L2$,

$$VEC[e2] = VEC[\psi_{k+1}(e1)] = \alpha_{f(e1)} . \tag{13}$$

From $\vdash E2$ we deduce at label $L1$:

$$\forall i(k < i \leqq G \wedge i \neq e1 \rightarrow VEC[\psi_k(i)] = \alpha_{f(i)}) . \tag{14}$$

Using property 4 we get at $L2$

$$\forall i(k < i \leqq G \wedge i \neq e1 \rightarrow VEC[\psi_k(i)] = VEC[\psi_{k+1}(i)] = \alpha_{f(i)}) . \tag{15}$$

Combining (13), (14) and (15) we have at $L2$:

$$\forall i(k+1 \leqq i \leqq G \rightarrow VEC[\psi_{k+1}(i)] = \alpha_{f(i)}) .$$

Passing from label $L2$ to label L $k := k+1$. Hence $\vdash E2$ at L for $k = k1+1$.

CASE 2. $kn=k$. In this case $\psi_k(k)=k$ and no exchange takes place.
From $\vdash E2$ at L and at $L1$ and $L2$ we deduce:

$$VEC[\psi_k(k)] = VEC[k] = \alpha_{f(k)} . \tag{16}$$

Combining (16) with $\vdash E1$ we get at $L2$

$$\forall i(1 \leqq i \leqq k \rightarrow VEC[i] = \alpha_{f(i)}) . \tag{17}$$

Hence

$$\forall i(1 \leqq i < k+1 \rightarrow VEC[i] = \alpha_{f(i)}) \text{ at } L2 . \tag{18}$$

From $\vdash E2$ and since there does not exist an element $e1 > k$ with $\psi_k(k)=e1$, and from property 4 it follows that:

$$\forall i(k+1 \leqq i \leqq G \rightarrow VEC[\psi_{k+1}(i)] = \alpha_{f(i)}) \text{ at } L2 . \tag{19}$$

Combining (18) and (19) at $L2$ and using the assignation $k := k+1$ in passing from label $L2$ to label L we get: $\vdash E1 \wedge E2$ at L for $k=k1+1$. By induction it follows that: $\vdash E1 \wedge E2$ at L for all $k=1(1)G+1$. Moreover $\vdash E1 \wedge E2 \wedge k=G+1$ at label Exh. In that case $E1$ confirms the truth of (1).

REMARK 1. The algorithm can be changed slightly in case of a matrix transposition. It suffices that the for loop runs from $k=2(1)G-2$, because $A[1,1]$ and $A[m,n]$ do not move. In case all elements have been moved up to $G-2$ then the $G-1$th element is in place. Even in the general case the range of the for loop can be taken $k=1(1)G-1$.

Remark 2. Looking at the invariant $\vdash E1 \wedge E2$ we observe that $E2$ describes the initial state of the program for $k=1$. $E1$ is then "empty". $E1$ describes the final state for $k=G+1$. $E2$ is then "empty".

Acknowledgements.

The author wishes to thank F. Göbel and J. Engelfriet for their discussions and remarks.

REFERENCES

1. P. F. Windley, *Transposing matrices in a digital computer*, Computer Journal 2 (April 1959, 47–48.
2. J. Bootroyd, *Algorithm 302, Transpose vector stored array*, Comm. ACM 10 (May 1967), 292–293.
3. S. Laflin and M. A. Brebner, *Algorithm 380, In situ-transposition of a rectangular matrix*, Comm. ACM 13 (May 1970), 324–326.

TECHNOLOGICAL UNIVERSITY TWENTE
ENSCHEDE
THE NETHERLANDS