1.3.3. Applications to Permutations

In this section we shall give several more examples of MIX programs, and at the same time introduce some important properties of permutations. These investigations will also bring out some interesting aspects of computer programming in general.

Permutations were discussed earlier in Section 1.2.5; we treated the permutation $c\,d\,f\,b\,e\,a$ as an *arrangement* of the six objects a, b, c, d, e, f in a straight line. Another viewpoint is also possible: We may think of a permutation as a *rearrangement* or renaming of the objects. With this interpretation it is customary to use a two-line notation, for example,

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}, \tag{1}$$

to mean "a becomes c, b becomes d, c becomes f, d becomes b, e becomes e, f becomes a." Considered as a rearrangement, this means that object c moves to the place formerly occupied by object a; considered as a renaming, it means that object a is renamed c. The two-line notation is unaffected by changes in the order of the columns; for example, the permutation (1) could also be written

$$\begin{pmatrix} c & d & f & b & a & e \\ f & b & a & d & c & e \end{pmatrix}$$

and in 718 other ways.

A *cycle notation* is often used in connection with this interpretation. Permutation (1) could be written

$$(a\,c\,f)\,(b\,d), \tag{2}$$

again meaning "a becomes c, c becomes f, f becomes a, b becomes d, d becomes b." A cycle $(x_1\,x_2\,\ldots\,x_n)$ means "x_1 becomes x_2, $\ldots$, x_{n-1} becomes x_n, x_n becomes x_1." Since e is fixed under the permutation, it does not appear in the cycle notation; that is, singleton cycles like "(e)" are conventionally not written. If a permutation fixes *all* elements, so that there are only singleton cycles present, it is called the *identity permutation*, and we denote it by "()".

The cycle notation is not unique. For example,

$$(b\,d)\,(a\,c\,f), \qquad (c\,f\,a)\,(b\,d), \qquad (d\,b)\,(f\,a\,c), \tag{3}$$

etc., are all equivalent to (2). However, "$(a\,f\,c)\,(b\,d)$" is not the same, since it says that a goes to f.

It is easy to see why the cycle notation is always possible. Starting with any element x_1, the permutation takes x_1 into x_2, say, and x_2 into x_3, etc., until finally (since there are only finitely many elements) we get to some element x_{n+1} that has already appeared among $x_1, \ldots, x_n$. Now x_{n+1} must equal x_1. For if it were equal to, say, x_3, we already know that x_2 goes into x_3; but by assumption, $x_n \neq x_2$ goes to x_{n+1}. So $x_{n+1} = x_1$, and we have a cycle $(x_1\,x_2\,\ldots\,x_n)$ as part of our permutation, for some $n \geq 1$. If this does not account for the entire permutation, we can find another element y_1 and get another cycle $(y_1\,y_2\,\ldots\,y_m)$

in the same way. None of the y's can equal any of the x's, since $x_i = y_j$ implies that $x_{i+1} = y_{j+1}$, etc., and we would ultimately find $x_k = y_1$ for some k, contradicting the choice of y_1. All cycles will eventually be found.

One application of these concepts to programming comes up whenever some set of n objects is to be put into a different order. If we want to rearrange the objects without moving them elsewhere, we must essentially follow the cycle structure. For example, to do the rearrangement (1), namely to set

$$(a, b, c, d, e, f) \leftarrow (c, d, f, b, e, a),$$

we would essentially follow the cycle structure (2) and successively set

$$t \leftarrow a, \quad a \leftarrow c, \quad c \leftarrow f, \quad f \leftarrow t; \quad t \leftarrow b, \quad b \leftarrow d, \quad d \leftarrow t.$$

It is frequently useful to realize that any such transformation takes place in disjoint cycles.

Products of permutations. We can multiply two permutations together, with the understanding that multiplication means the application of one permutation after the other. For example, if permutation (1) is followed by the permutation

$$\begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix},$$

we have a becomes c, which then becomes c; b becomes d, which becomes a; etc.:

$$\begin{aligned} \begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \times \begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix} &= \begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \times \begin{pmatrix} c & d & f & b & e & a \\ c & a & e & d & f & b \end{pmatrix} \\ &= \begin{pmatrix} a & b & c & d & e & f \\ c & a & e & d & f & b \end{pmatrix}. \end{aligned} \tag{4}$$

It should be clear that multiplication of permutations is not commutative; in other words, $\pi_1 \times \pi_2$ is not necessarily equal to $\pi_2 \times \pi_1$ when π_1 and π_2 are permutations. The reader may verify that the product in (4) gives a different result if the two factors are interchanged (see exercise 3).

Some people multiply permutations from right to left rather than the somewhat more natural left-to-right order shown in (4). In fact, mathematicians are divided into two camps in this regard; should the result of applying transformation T_1, then T_2, be denoted by T_1T_2 or by T_2T_1? Here we use T_1T_2.

Equation (4) would be written as follows, using the cycle notation:

$$(a\,c\,f)\,(b\,d)\,(a\,b\,d)\,(e\,f) = (a\,c\,e\,f\,b). \tag{5}$$

Note that the multiplication sign "$\times$" is conventionally dropped; this does not conflict with the cycle notation since it is easy to see that the permutation $(a\,c\,f)(b\,d)$ is really the product of the permutations $(a\,c\,f)$ and $(b\,d)$.

Multiplication of permutations can be done directly in terms of the cycle notation. For example, to compute the product of several permutations

$$(a\,c\,f\,g)\,(b\,c\,d)\,(a\,e\,d)\,(f\,a\,d\,e)(b\,g\,f\,a\,e), \tag{6}$$

we find (proceeding from left to right) that "a goes to c, then c goes to d, then d goes to a, then a goes to d, then d is unchanged"; so the net result is that a goes to d under (6), and we write down "$(a\,d$" as the partial answer. Now we consider the effect on d: "d goes to b goes to g"; we have the partial result "$(a\,d\,g$". Considering g, we find that "g goes to a, to e, to f, to a", and so the first cycle is closed: "$(a\,d\,g)$". Now we pick a new element that hasn't appeared yet, say c; we find that c goes to e, and the reader may verify that ultimately the answer "$(a\,d\,g)(c\,e\,b)$" is obtained for (6).

Let us now try to do this process by computer. The following algorithm formalizes the method described in the preceding paragraph, in a way that is amenable to machine calculation.

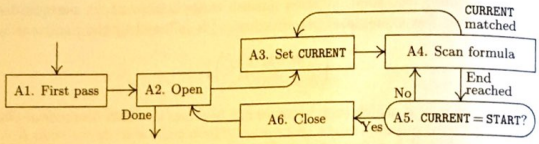

Fig. 20. Algorithm A for multiplying permutations.

Algorithm A (*Multiply permutations in cycle form*). This algorithm takes a product of cycles, such as (6), and computes the resulting permutation in the form of a product of disjoint cycles. For simplicity, the removal of singleton cycles is not described here; that would be a fairly simple extension of the algorithm. As this algorithm is performed, we successively "tag" the elements of the input formula; that is, we mark somehow those symbols of the input formula that have been processed.

A1. [First pass.] Tag all left parentheses, and replace each right parenthesis by a tagged copy of the element that follows its matching left parenthesis. (See the example in Table 1.)

A2. [Open.] Searching from left to right, find the first untagged element of the input. (If all elements are tagged, the algorithm terminates.) Set START equal to it; output a left parenthesis; output the element; and tag it.

A3. [See CURRENT.] Set CURRENT equal to the next element of the formula.

A4. [Scan formula.] Proceed to the right until either reaching the end of the formula, or finding an element equal to CURRENT; in the latter case, tag it and go back to step A3.

Table 1

ALGORITHM A APPLIED TO (6)

After step	START	CURRENT		
A1			(a c f g a (b c d b (a e d a (f a d e f (b g f a e b	Output
A2	a		(a̲ c f g a (b c d b (a e d a (f a d e f (b g f a e b	
A3	a	c	(a c̲ f g a (b c d b (a e d a (f a d e f (b g f a e b	(a
A4	a	c	(a c f g a (b c̲ d b (a e d a (f a d e f (b g f a e b	
A4	a	d	(a c f g a (b c d̲ b (a e d a (f a d e f (b g f a e b	
A4	a	a	(a c f g a (b c d b (a e d a̲ (f a d e f (b g f a e b	
A5	a	d	(a c f g a (b c d b (a e d a (f a̲ d e f (b g f a e b	
A5	a	g	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b̲	d
A5	a	a	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b̲	g
A6	a	a	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b̲	)
A2	c	a	(a c̲ f g a (b c d b (a e d a (f a d e f (b g f a e b	(c
A5	c	e	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b̲	e
A5	c	b	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b̲	b
A6	c	c	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b̲	)
A6	f	f	(a c f g a (b c d b (a e d a (f a d e f (b g f a e b̲	(f)

Here ̲ represents a cursor following the element just scanned; tagged elements are light gray.

A5. [CURRENT = START?] If CURRENT ≠ START, output CURRENT and go back to step A4 starting again at the left of the formula (thereby continuing the development of a cycle in the output).

A6. [Close.] (A complete cycle in the output has been found.) Output a right parenthesis, and go back to step A2. ∎

For example, consider formula (6); Table 1 shows successive stages in its processing. The first line of that table shows the formula after right parentheses have been replaced by the leading element of the corresponding cycle; succeeding lines show the progress that is made as more and more elements are tagged. A cursor shows the current point of interest in the formula. The output is "$(a\,d\,g)(c\,e\,b)(f)$"; notice that singleton cycles will appear in the output.

A MIX program. To implement this algorithm for MIX, the "tagging" can be done by using the sign of a word. Suppose our input is punched onto cards in the following format: An 80-column card is divided into 16 five-character fields. Each field is either (a) "␣␣␣␣(", representing the left parenthesis beginning a cycle; (b) ")␣␣␣␣", representing the right parenthesis ending a cycle; (c) "␣␣␣␣␣", all blanks, which may be inserted anywhere to fill space; or (d) anything else, representing an element to be permuted. The last card of the input is recognized by having columns 76–80 equal to "␣␣␣␣=". For example, (6) might be punched