

An Example of How to Flip an Array

If you are not familiar with the topic of arrays, first read the web paged entitled [Array Declaration and Manipulation](#).

Analysts and programmers often visualize storage arrays as vertical stacks of memory locations, all with a common identifier (such as N) and each with a unique referencing number called a "subscript". Many people visualize and draw arrays as vertical tables, with the first (lowest numbered) element at the [bottom](#) and the last (highest numbered) element at the [top](#). In the example below for an array of 4 elements, we would refer to the element containing the 20 value as the "top of the array".

		N		
Highest subscript →	3	20	←	Top element
	2	40		
	1	10		
Lowest subscript →	0	30	←	Bottom element

The best way to learn how to manipulate arrays is to practice moving or shuffling the values in their elements to different positions. For example, we might try to "flip" the array pictured above such that the 20 value (the top element) was swapped with the 30 (the bottom element) value and the 40 value was swapped with the 10 value, and so on, until all of the array values were repositioned with their counterparts at the opposite end of the array. Stop and think about this for a moment and see if you could imagine an algorithm to do this. Then read on.

The flip procedure could be done the hard way, with many C++ statements like this one:

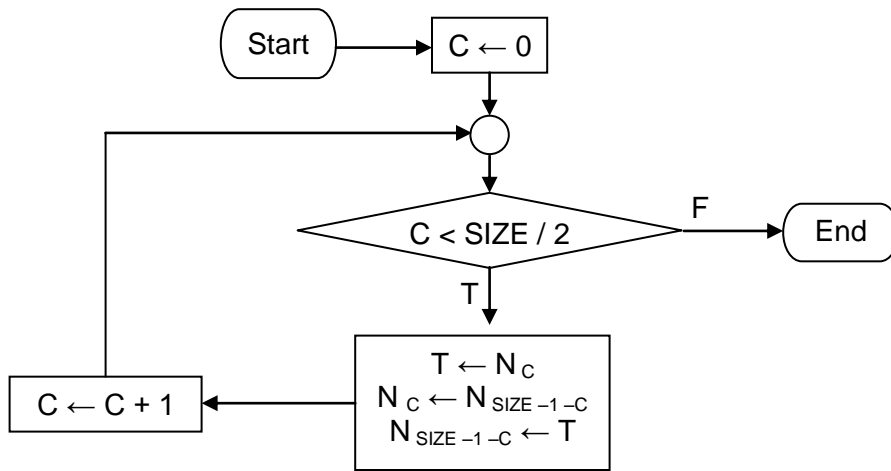
```
N[0] = N[3];
```

The problem with the statement above is that it will overwrite the value 30 at N[0] with the 20 from N[3]. The solution is to use a "swap variable" to preserve the original value until it can be written back to its position, as shown below with the set of statements:

```
T = N[0];    /* Preserve the value from one element */
N[0] = N[3]; /* Overwrite the preserved element with its opposite element */
N[3] = T;    /* Write the retained value to the opposite element */
```

This works, but would require a multitude of statements, especially if the array had many elements. A better solution is to place the statements above into the body of a counting loop and replace the constant subscripts (0 and 3) with variables or expressions (formulas) that could relate to the value of the counter (and to the size of the array). The counter could start with a value of 0 and then be incremented after every pass until enough of the pairs of elements had been swapped. The 0 in the statement above would be replaced with the label for the counter (perhaps C) and the 3 above would be replaced with (can you figure it out?). The new subscript should be related to the size of the array. If we use a symbolic constant such as SIZE for that value (4 in the array pictured above), the new subscript replacing the 3 in the statements above should be the expression: SIZE-1-C.

We can visualize the algorithm for the flipping procedure as a flowchart as shown below. Notice the standard subscript notation used for subscripts in logic documents. For example, the element that would be written in C++ as $N[0]$ would be written normally as N_0 in a flowchart or algorithm outline.



Notice the condition in the diamond. Can you tell why the counter must stay below a value determined by the expression $SIZE-2$? This is because the swapping statements affect a pair of elements, one pointed to by the counter (C), the other on the opposite end of the array. If we did not stop the counter when it was pointing at the middle of the array, we would accidentally swap all of the elements back to their original positions. The flowchart above would be just the middle part of an overall flowchart that loaded the array and later displayed its elements after the flip was complete. The data used in the flowchart above would be documented as follows:

SYMBOLIC CONSTANT LIST

LABEL	DESCRIPTION	VALUE	USAGE	DESTINATION
SIZE	Quantity of elements in the array	4	Tested	---

VARIABLE LIST

LABEL	DESCRIPTION	SOURCE	USAGE	DESTINATION
ARRAYS:				
N	Group of whole numbers	Given *	Tested and swapped using T	Screen
SCALARS:				
C	General purpose counter	Set to 0	Loop index	---
T	Swap variable	Assigned	For N	---

* - "Given" means that the values in the array are provide in the variable declaration statement.

You could write the code for this procedure in C++ as shown on the page below. The portion in ***bold italics*** represents the flipping procedure. The remaining code is provided to supply a complete program that can be tested. You could try many other manipulations of the given array by developing your own objectives, algorithm/flowchart, and then revising the italicized code below to match your algorithm.

```

/*****
 * practice.cpp = Flips the contents of an integer array *
 * Written by Randy Gibson - October 1, 2011 *
 *****/

#include <iostream> // Load the Input/Output STREAM Library
using namespace std; // to define context for cin and cout
#define SIZE 4 // Size of the array

int main ()
{
    /* Arrays */
    int N[]={30,10,40,20}; // An array of known integers

    /* Scalars */
    int C, // General purpose counter
    T; // Temporary variable for swapping

    cout << "ARRAY MANIPULATOR\n";
    cout << "by Randy Gibson - October 1, 2011\n\n";

    cout << "Array order before flip:\n\n";
    for (C=0; C<SIZE; C=C+1) cout << N[C] << endl;

    for (C=0; C<SIZE/2; C=C+1)
    {
        T = N[C];
        N[C] = N[SIZE-1-C];
        N[SIZE-1-C] = T;
    }

    cout << "\nArray order after flip:\n\n";
    for (C=0; C<SIZE; C=C+1) cout << N[C] << endl;

    return 0;
}

```