UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

Swapping between arrays

Douglas Wilhelm Harder, M.Math. LEL
Prof. Hiren Patel, Ph.D., P.Eng.
Prof. Werner Dietl, Ph.D.

© 2018 by Douglas Wilhelm Harder and Hiren Patel.
Some rights reserved.

BY NC SA

Outline

- In this lesson, we will:
  - Discuss swapping a sequence of entries between two arrays
  - Describe the number of assignments
  - Determine if we can reduce the number of assignments if the order of the swaps does not matter

BY NC SA

Swapping between arrays

- Suppose you want to swap m entries between two arrays,
  the first starting at position n_1 and
  the second starting at position n_2
  - You may assume both arrays are sufficiently large

- The function declaration would be:

```
void swap( double array1[], std::size_t n1,
           double array2[], std::size_t n2,
           std::size_t m );
```

BY NC SA

Swapping between arrays

- This should be a simple for loop that has *m* swaps:

```
void swap( double array1[], std::size_t n1,
           double array2[], std::size_t n2,
           std::size_t m ) {
    for ( std::size_t k{ 0 }; k < m; ++k ) {
        double tmp{ array1[n1 + k] };
        array1[n1 + k] = array2[n2 + k];
        array2[n2 + k] = tmp;
    }
}
```

cc BY NC SA

Number of assignments

- This requires 3*m* assignments
  - One of these, at each step, is the initialization of `tmp`

- Question: Can we do better if we don't care about the order?
  - For example,
    suppose this is just related data that we want to rearrange
  - Can we get it down to 2*m* assignments?
  - Pause and think about this

Swapping entries

- For example, suppose we have the following:

```
swap( a1, 1, a2, 3, 4 );
```

a1

3.2	4.5	1.2	8.5	6.2	4.9	7.0	9.3	0.6

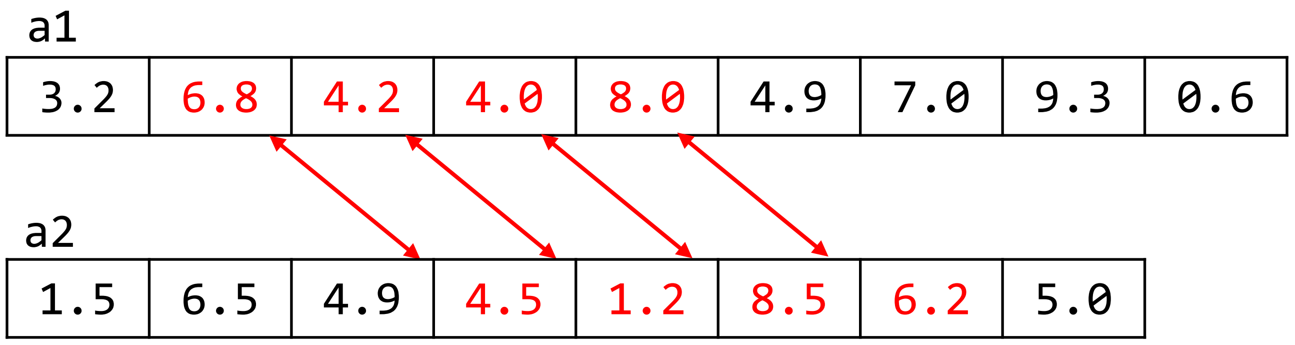
a2

1.5	6.5	4.9	6.8	4.2	4.0	8.0	5.0

Swapping entries

- All we can do is swap the entries:

```
swap( a1, 1, a2, 3, 4 );
```



BY NC SA

Swapping without concern for order

- Suppose we don't care about how they are swapped

```
unordered_swap( a1, 1, a2, 3, 4 );
```

a1

3.2	4.5	1.2	8.5	6.2	4.9	7.0	9.3	0.6

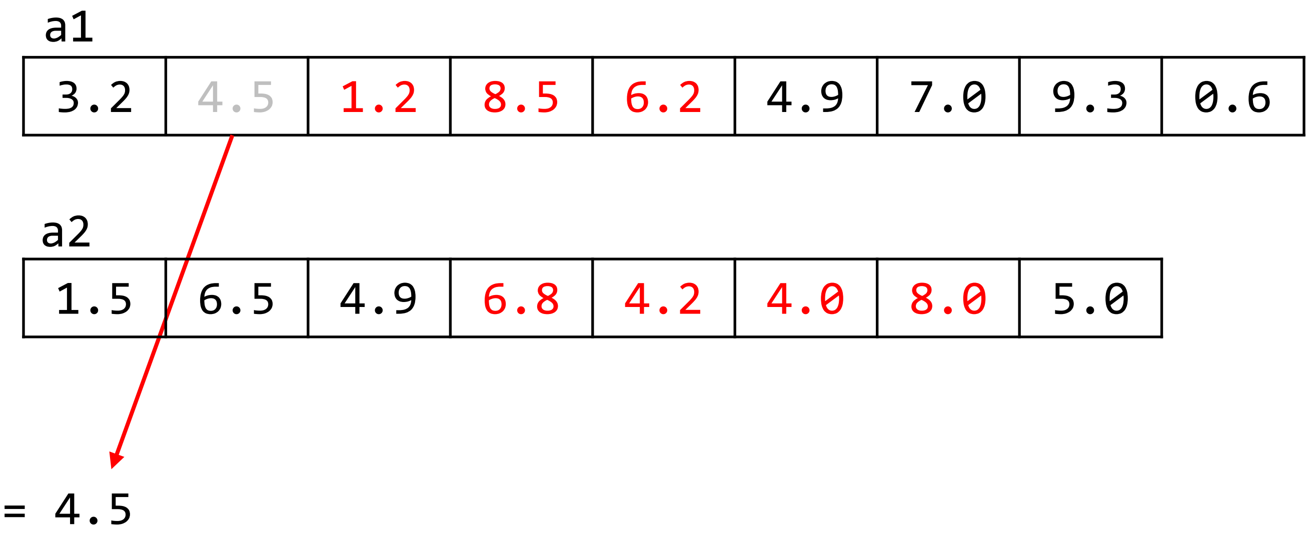
a2

1.5	6.5	4.9	6.8	4.2	4.0	8.0	5.0

BY NC SA

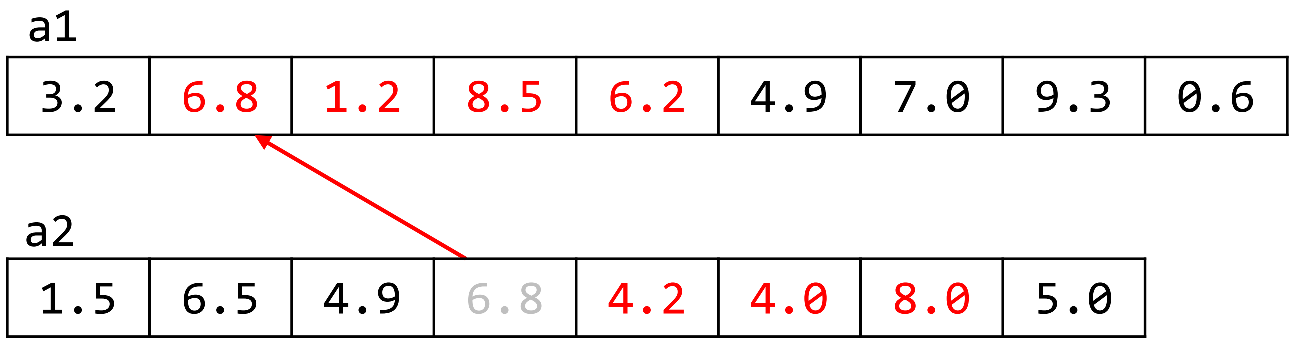
Swapping without concern for order

- Assign the first entry in one array to a temporary variable



Swapping without concern for order

- This creates a *hole*, so fill it with the first entry of the next array

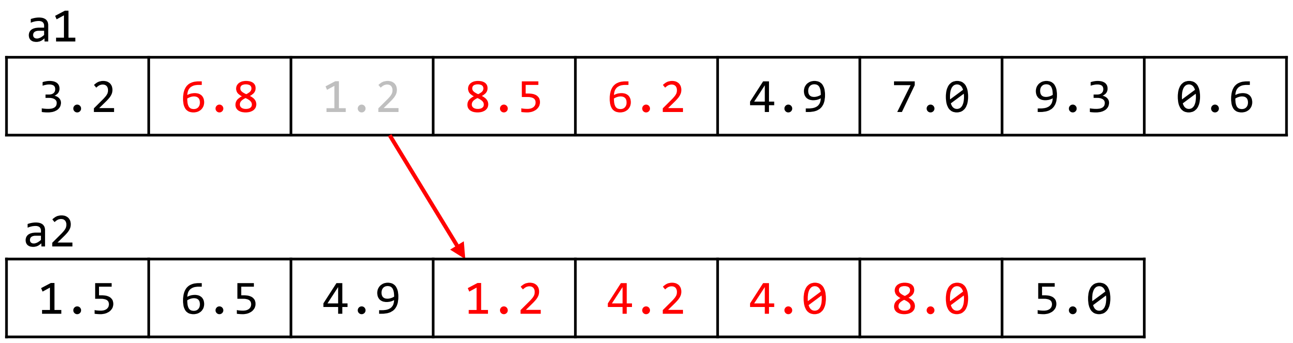


```
tmp = 4.5
```

BY NC SA

Initial approach

- Now fill this hole with the second entry of the first array, and so on



```
tmp = 4.5
```

cc BY NC SA

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

Swapping without concern for order

- After 2*m* assignments, there is a hole at the end of the second array

a1

3.2	6.8	4.2	4.0	8.0	4.9	7.0	9.3	0.6

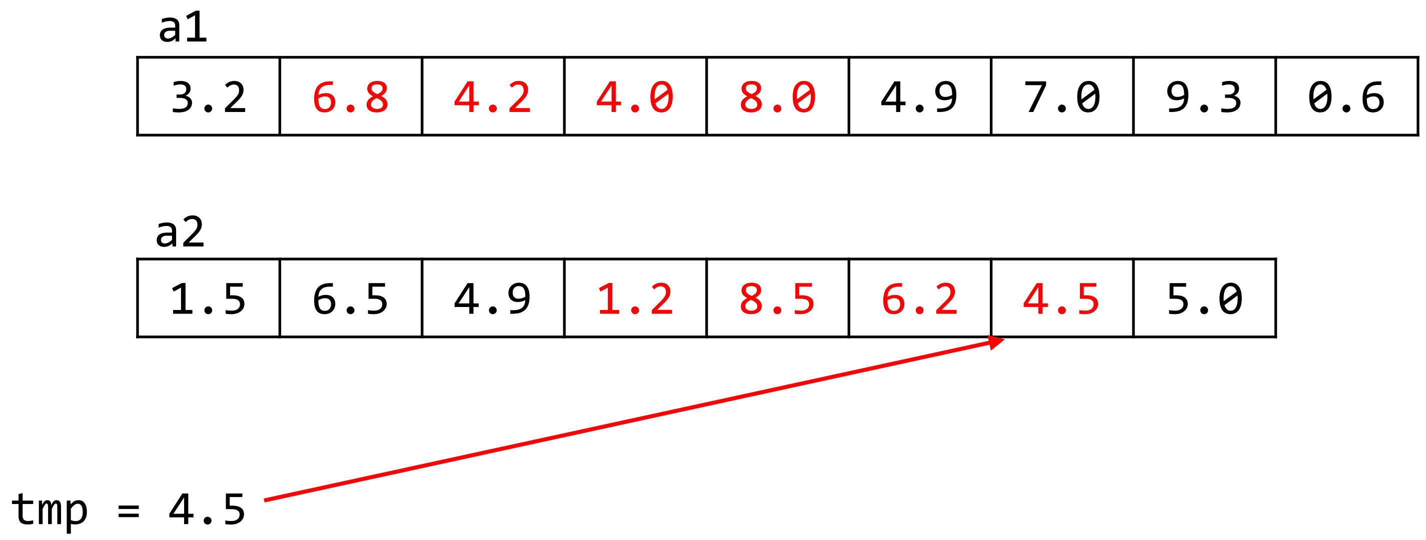
a2

1.5	6.5	4.9	1.2	8.5	6.2	8.0	5.0

```
tmp = 4.5
```

Swapping without concern for order

- After 2*m* assignments, there is a hole at the end of the second array
  - Fill this with the temporarily stored value



cc BY NC SA

Implementation

- The code is slightly more complex, as we must be careful:

```
void unordered_swap( double array1[], std::size_t n1,
                     double array2[], std::size_t n2,
                     std::size_t m ) {
    if ( m == 0 ) {
        return ;
    }

    double tmp{ array1[n1] };

    for ( std::size_t k{ 0 }; k < m - 1; ++k ) {
        array1[n1 + k] = array2[n2 + k];
        array2[n2 + k] = array1[n1 + k + 1];
    }

    array1[n1 + m - 1] = array2[n2 + m - 1];
    array2[n2 + m - 1] = tmp;
}
```

BY NC SA

Comparison

- We may now observe that:
  - The original code required $3m$ assignments
  - If we don't care about swaps, we can reduce this to $2m + 1$

- The entries from the first array are rotated by one in the second

BY NC SA

Summary

- Following this lesson, you now
  - Know that swapping entries between arrays can be optimized
  - Understand we can reduce the number of assignments by almost one third if we don't care about the order of the copies

BY NC SA

References

[1] https://en.wikipedia.org/wiki/Array_data_structure

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.

BY NC SA