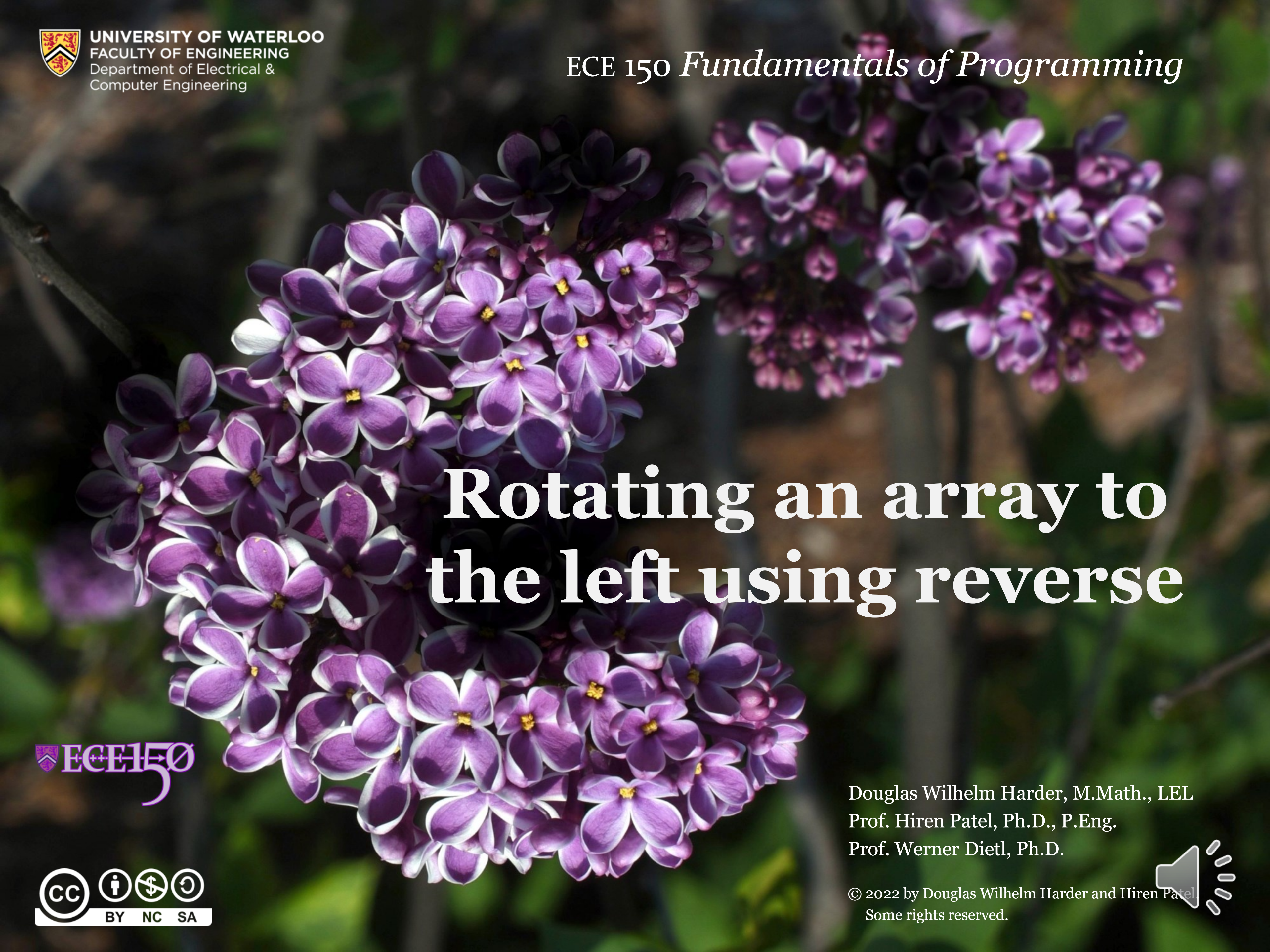


UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 150 *Fundamentals of Programming*

Rotating an array to the left using reverse

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

Outline

- In this lesson, we will:
  - Describe a rotation of an array to the left
  - Look at three implementations
    - Each will be more successively more efficient
    - The first two will require temporary arrays
    - The last will not
  - Discuss the need to look for solutions to problems and not just implement the first solution that comes to your mind

Introduction

- Rotating an array to the left by k is the process of moving each entry k locations to the left of its original index, with the first k entries being moved to the end
- For example, given the array:

6.7	-0.5	-5.5	-8.4	4.4	1.1	5.1	-1.4	7.3	7.4	9.3

  - Rotating to the left by 3 results in

-8.4	4.4	1.1	5.1	-1.4	7.3	7.4	9.3	6.7	-0.5	-5.5

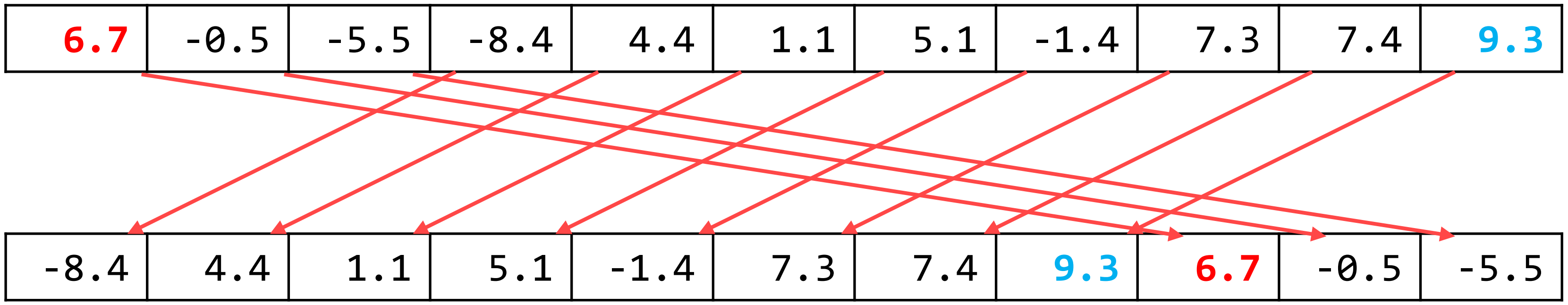
  - Rotating the original array to the left by 9 moves results in

7.4	9.3	6.7	-0.5	-5.5	-8.4	4.4	1.1	5.1	-1.4	7.3

    - This is the same as rotating the original array by -2
    - This is also described as rotating to the right by 2

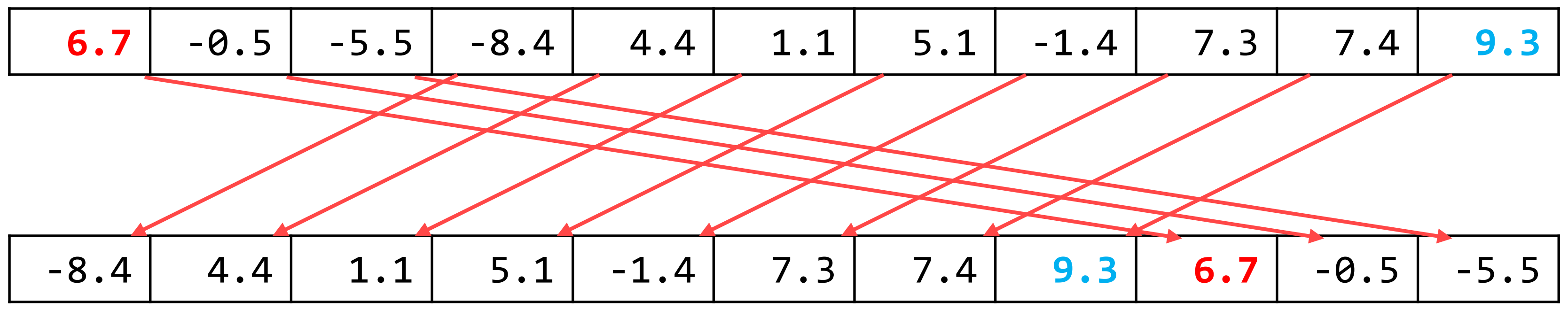
Introduction

- Rotating by 3 requires the following assignments:



Introduction

- How can we implement this?



6.7	-0.5	-5.5	-8.4	4.4	1.1	5.1	-1.4	7.3	7.4	9.3

-8.4	4.4	1.1	5.1	-1.4	7.3	7.4	9.3	6.7	-0.5	-5.5

- The easiest solution is to:
  - Create a second array
  - Copy all the entries over
  - Copy all the entries back to their new location

Initial implementation

```
void rotate_array( double array[], std::size_t capacity, std::size_t rotation ) {
    rotation = rotation%capacity;

    if ( rotation == 0 ) {
        return;
    }

    double a_tmp[capacity];

    for ( std::size_t k{ 0 }; k < capacity; ++k ) {
        a_tmp[k] = array[k];
    }

    for ( std::size_t k{ 0 }, posn{ capacity - rotation }; k < capacity; ++k ) {
        array[posn] = a_tmp[k];
        ++posn;

        if ( posn == capacity ) {
            posn = 0;
        }
    }
}
```

An incremental improvement

- To rotate to the left by *k* we could, instead,
  - Copy the first *k* entries to the temporary array
  - Copy the remaining entries to the start of the original array
  - Copy the *k* entries back to the end of the original array

- This at least results in fewer assignments
  - With care, we can ensure no more than half the entries are copied to the temporary array

Second implementation

```
void rotate_array( double array[], std::size_t capacity, std::size_t rotation ) {
    rotation = rotation%capacity;

    if ( rotation == 0 ) {
        return;
    }

    double a_tmp[rotation];

    for ( std::size_t k{ 0 }; k < rotation; ++k ) {
        a_tmp[k] = array[k];
    }

    std::size_t posn{ 0 };

    for ( std::size_t k{ rotation }; k < capacity; ++k, ++posn ) {
        array[posn] = array[k];
    }

    for ( std::size_t k{ 0 }; k < rotation; ++k, ++posn ) {
        array[posn] = a_tmp[k];
    }
}
```

Issues

- Problem:
  - We must allocate a temporary array and
    - Copy entries to that temporary array, and
    - Copy entries back into the initial array

- This is potentially expensive…
  - Can we do this without a temporary array?

Rotating using reversals

- Suppose we want to rotate the entries by 3:

6.7	-0.5	-5.5	-8.4	4.4	1.1	5.1	-1.4	7.3	7.4	9.3

-5.5	-0.5	6.7

-8.4	4.4	1.1	5.1	-1.4	7.3	7.4	9.3

-5.5	-0.5	6.7	9.3	7.4	7.3	-1.4	5.1	1.1	4.4	-8.4

-8.4	4.4	1.1	5.1	-1.4	7.3	7.4	9.3	6.7	-0.5	-5.5

- Thus, to rotate to the left by 3:
  - Reverse the first 3 entries
  - Reverse the last $n - 3$ entries
  - Reverse all the entries

Reversing an array

- Recall the reverse function

```
void swap_array_entries( double array[], std::size_t m, std::size_t n ) {
    double tmp{ array[m] };
    array[m] = array[n];
    array[n] = tmp;
}

void reverse_array( double array[], std::size_t capacity ) {
    for ( std::size_t first{ 0 }, second{ capacity - 1 };
          first < second; ++first, --second ) {
        swap_array_entries( array, first, second );
    }
}
```

Reversing part of an array

- We can generalize this to reverse the entries between indices m and $n - 1$:

```
void reverse_array( double array[], std::size_t m, std::size_t n ) {
    for ( std::size_t first{ m }, second{ n - 1 };
          first < second; ++first, --second ) {
        swap_array_entries( array, first, second );
    }
}
```

Third implementation

- Here is a nicer implementation:

```
void rotate_array( double array[], std::size_t capacity, std::size_t rotation ) {
    rotation = rotation%capacity;

    if ( rotation == 0 ) {
        return;
    }

    reverse_array( array, 0, rotation );
    reverse_array( array, rotation, capacity );
    reverse_array( array, 0, capacity );
}
```

Thoughts…

- Could you have determined this?
  - Probably not…
- Always remember:
  - If you're working on a problem,
    someone may have come up with a much better solution already
  - There are often interesting and ingenious solutions out there

Summary

- Following this lesson, you now:
  - Know what a rotation is
  - Have seen a few implementations using temporary arrays
  - Understand these require additional memory which may not always be available
  - Have seen how we can use an array reversing function to implement a rotation

References

[1] https://en.wikipedia.org/wiki/Circular_shift
[2] https://en.cppreference.com/w/cpp/algorithm/rotate

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