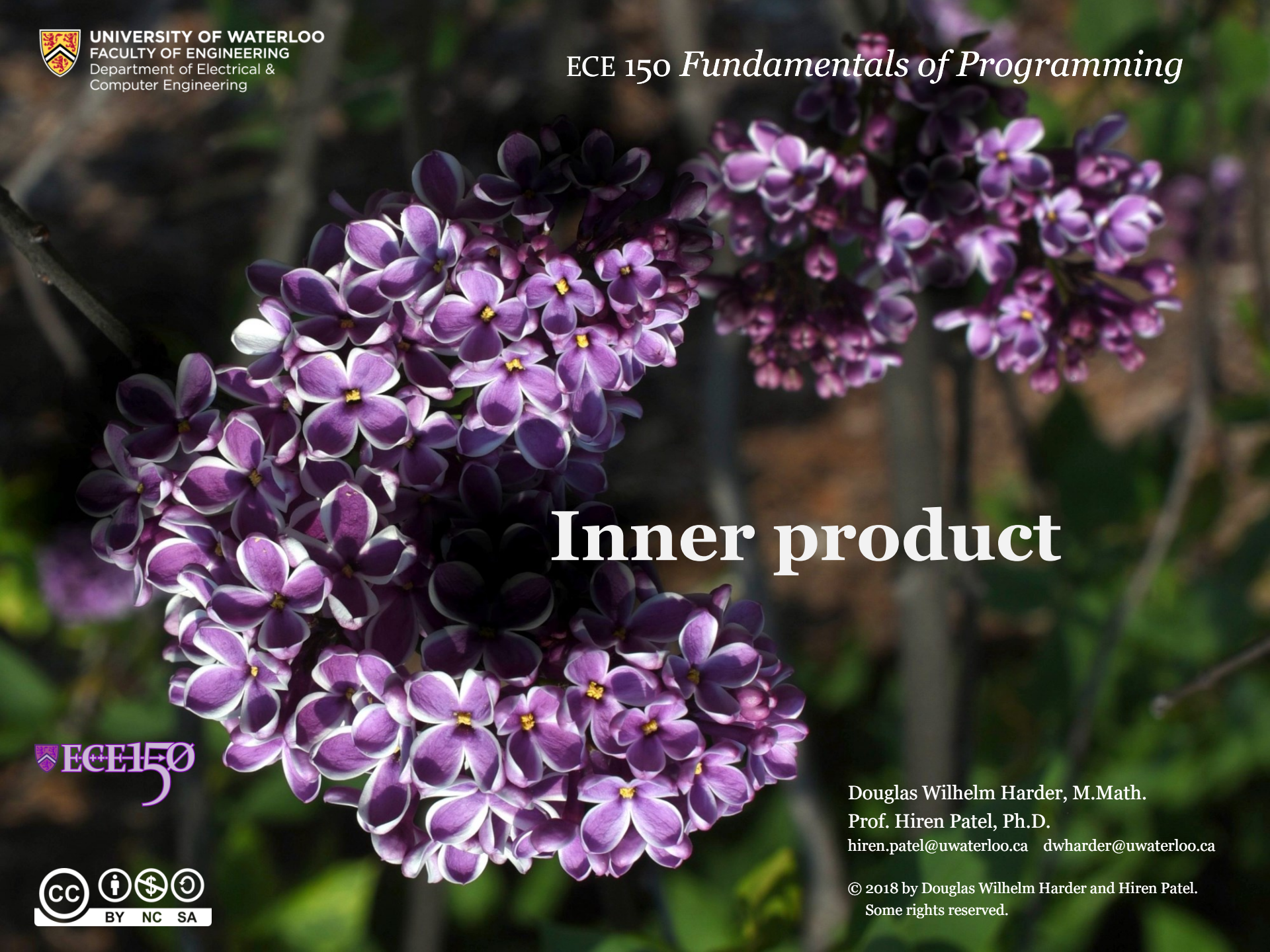
UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*



# Inner product

ECE150

Douglas Wilhelm Harder, M.Math.
Prof. Hiren Patel, Ph.D.
hiren.patel@uwaterloo.ca dwharder@uwaterloo.ca

# Outline

- In this lesson, we will:
  - Review the inner product
  - Implement it in C++
  - Consider how to generalize the ranges
  - Consider how to generalize the functions

# Introduction

- In linear algebra,
  you have already seen the inner product of two vectors:
  - The sum of the pairwise products of the entries
  - We could implement this in C++ for arguments that are arrays

# Implementing the inner product

- For example,

```
double inner_product( double array1[],
                      double array2[],
                      std::size_t capacity ) {
    double result{ 0.0 };

    for ( std::size_t k{0}; k < capacity; ++k ) {
        result += array1[k]*array2[k];
    }

    return result;
}
```

# Generalizing the range

- We could, however, allow the inner product to be taken along arbitrary ranges of the arrays:

```
double inner_product(
    double array1[], std::size_t begin1, std::size_t end1,
    double array2[], std::size_t begin2
) {
    double result{ 0.0 };

    for ( std::size_t k1{begin1}, k2{begin2}; k1 < end1; ++k1, ++k2 ) {
        result += array1[k1]*array2[k2];
    }

    return result;
}
```

If will assume `array2[k2]` is defined for the same width as for `array1`

# Generalizing the operations

- Suppose, however, we wanted a different pair-wise operation, and a different means of collating the information from these results

```
double inner_product(
    double array1[], std::size_t begin1, std::size_t end1,
    double array2[], std::size_t begin2
) {
    double result{ 0.0 };

    for ( std::size_t k1{begin1}, k2{begin2}; k1 < end1; ++k1, ++k2 ) {
        result += array1[k1]*array2[k2];
    }

    return result;
}
```

  - Recall, however, that the choice of using pairwise multiplication and summing the results is arbitrary

# Generalizing the operations

- We could let the user pass two bivariate functions

```
double sum( double x, double y ) {
    return x + y;
}

double product( double x, double y ) {
    return x*y;
}
```

# Generalizing the operations

- We would then call these functions:

```
double inner_product(
    double array1[], std::size_t begin1, std::size_t end1,
    double array2[], std::size_t begin2,
    std::function<double( double, double )> sum,
    std::function<double( double, double )> product
) {
    double result{ 0.0 };

    for ( std::size_t k1{begin1}, k2{begin2}; k1 < end1; ++k1, ++k2 ) {
        result = sum( result, product( array1[k1], array2[k2] ) );
    }

    return result;
}
```

# Generalizing the operations

- We'd probably want an initial value, too:

```
double inner_product(
    double array1[], std::size_t begin1, std::size_t end1,
    double array2[], std::size_t begin2,
    double x0,
    std::function<double( double, double )> sum,
    std::function<double( double, double )> product
) {
    double result{ x0 };

    for ( std::size_t k1{begin1}, k2{begin2}; k1 < end1; ++k1, ++k2 ) {
        result = sum( result, product( array1[k1], array2[k2] ) );
    }

    return result;
}
```

# Example 1

- Now, to call the inner product, we would use

```
int main() {
    std::size_t N{ 5 };
    double vector1[N]{ 3.2, -5.4,  1.9,  8.6,  0.7 };
    double vector2[N]{ 6.5,  2.0,  7.1, -4.3, -9.8 };

    std::cout << inner_product( vector1, 0, N, vector2, 0, 0.0,
                                sum, product ) << std::endl;

    return 0;
}

double sum( double x, double y ) {
    return x + y;
}

double product( double x, double y ) {
    return x*y;
}
```

# Example 2

- What does this code do?

```
int main() {
    std::size_t N{ 5 };
    double vector1[N]{ -0.3, -0.2, -0.1,  0.0,  0.1 };
    double vector2[N]{  0.9,  0.4,  0.1,  0.0,  0.1 };

    std::cout << inner_product( vector1, 0, N, vector2, 0, 0.0,
                                sum, equals ) << std::endl;

    return 0;
}

double sum( double x, double y ) {
    return x + y;
}

double equals( double x, double y ) {
    return (x == y);
}
```

# Example 3

- What does this code do?

```
int main() {
    std::size_t N{ 5 };
    double vector1[N]{ -0.3, -0.2, -0.1,  0.0,  0.1 };
    double vector2[N]{  0.9,  0.4,  0.1,  0.0,  0.1 };

    std::cout << inner_product( vector1, 0, N, vector2, 0,
                                -std::numeric_limits<double>::infinity(),
                                 max, abs_sum ) << std::endl;

    return 0;
}

double max( double x, double y ) {
    if ( x >= y ) {
        return x;
    } else {
        return y;
    }
}

double abs_sum( double x, double y ) {
    return std::abs( x ) + std::abs( y );
}
```

# The standard library

- In the standard library, there is a

  `std::inner_product(…)`

  in the header

  `#include <numeric>`

  - Again, despite it appearing there are many function evaluations, a good compiler will eliminate these and simply inline these operations
  - Rather than passing an array pointer and indices, you pass the addresses of `array[begin]` and `array[end]`

# Summary

- Following this lesson, you now:
  - Have reviewed the inner product
  - Have seen an implementation
  - Know how to generalizing the ranges
  - Understand how to generalize the operations and use this

# References

[1] https://www.cplusplus.com/reference/numeric/inner_product/
[2] https://en.cppreference.com/w/cpp/algorithm/inner_product

# 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.