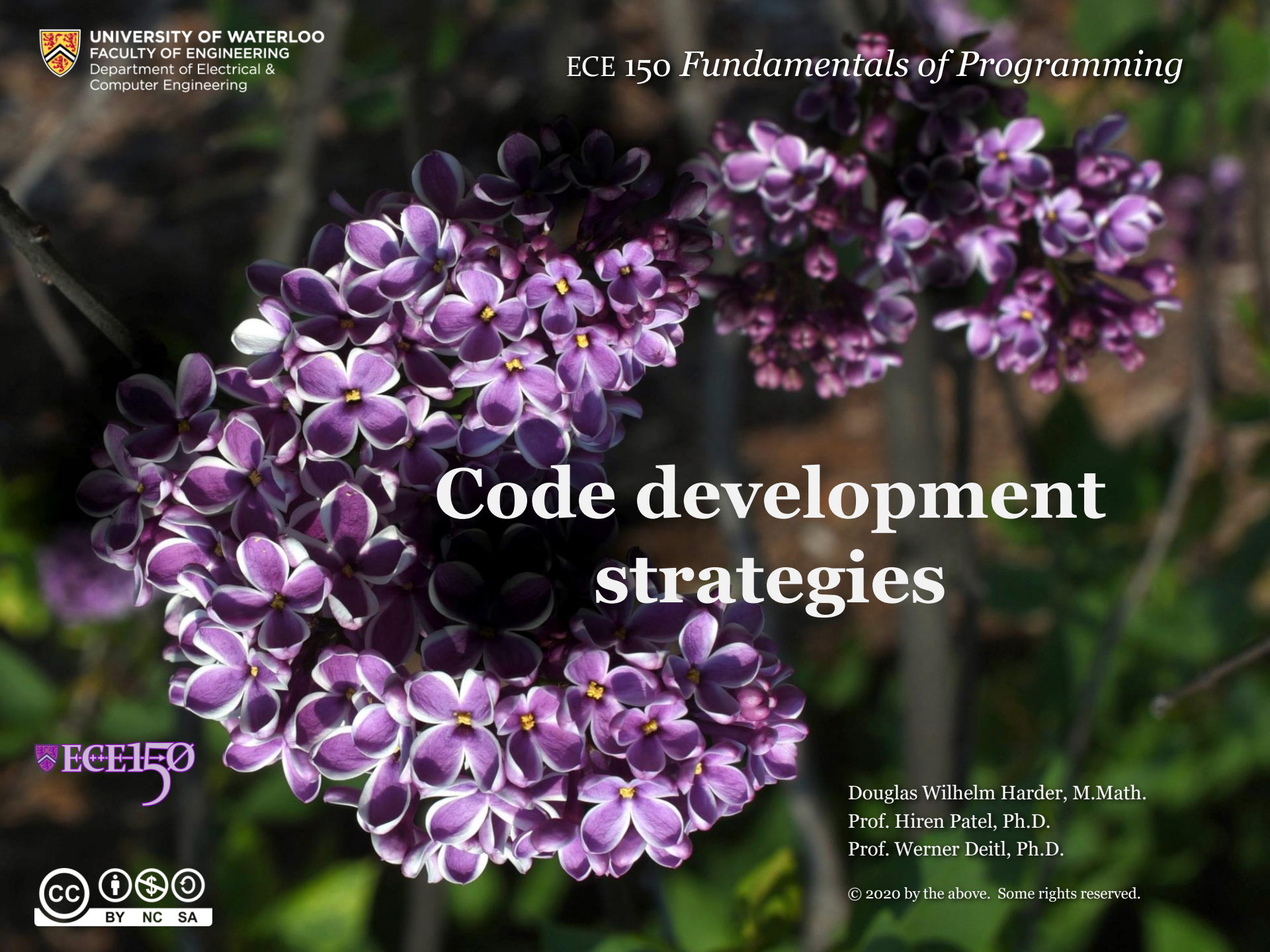
UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*



# Code development strategies

ECE150

Douglas Wilhelm Harder, M.Math.
Prof. Hiren Patel, Ph.D.
Prof. Werner Deitl, Ph.D.

# Outline

- This is the second in a sequence of six topics on
  - C assertions
  - Code development strategies
  - Testing
  - Commenting your code
  - Using print statements for debugging
  - Using tracing for debugging

# Outline

- In this tutorial, we will:
  - Describe how to start a project
  - Using a skeleton framework
  - Deal with assertions on the parameters
  - Consider implementing helper functions

# Code skeletons

- Every project will involve you authoring various functions
  - The project specifications will describe:
    - The parameters and their types
    - The return type
    - The relationship between the parameters and what is returned
      - That is, what is the function supposed to accomplish

- We will describe an approach that will be most beneficial
  - We will build a skeleton for each function
  - We will then check the parameters
  - You will then author each function one at a time

# Code skeletons

- The wrong approach to take is to:
  - First write all of the functions
  - Try to compile once you think everything is finished

# Code skeletons

- Instead, suppose you were asked to author:

```
int factorial( int n );
int binomial( int n, int k );
```

# Code skeletons

- Start with:

```
#include <iostream>

// Function declarations
int main();
int factorial( int n );
int binomial( int n, int k );

// Function definitions
int main() {
    return 0;
}

int factorial( int n ) {
    return 0;
}

int binomial( int n, int k );
    return 0;
}
```

# Code skeletons

- In each case, we returned a default value: `0`
- The most wonderful aspect of this code is:

  **It compiles!**

  - The next step is to implement and test one function at a time!
  - Always make sure one function works before you start the next

# Assertions on parameters

- The next step is to add any assertions on the parameters:

```
int factorial( int n ) {
    assert( n >= 0 );
    return 0;
}
```

# Helper functions

- Next, you can immediately think of additional functions you could use in authoring your code,

  add them to your declarations and skeletons:

```
// Function declarations
int main();
int factorial( int n );
int binomial( int n, int k );
int falling_factorial( int n, int k );

// Function definitions
//    ...insert other function definitions here...

int falling_factorial( int n, int k ) {
    assert( k >= 0 );
    return 0;
}
```

# Helper functions

- You may always, at all times,
  add any additional functions you may feel will simplify your life

- If you every cut-and-paste code, always ask yourself if it would be better to split off the code and write a function

You never need to ask
"Can I write my own function?"
"Can I add another function to my file?"

Just be sure to submit it!

# Another example

- Suppose you were asked to author both GCD and LCM functions

```
#include <iostream>

// Function declarations
int main();
int gcd( int m, int n );
int lcm( int m, int n );

// Function definitions
int main() {
    return 0;
}

int gcd( int m, int n ) {
    return 0;
}

int lcm( int m, int n ) {
    return 0;
}
```

# Simple parameter operations

- Next, you may deduce from the project description that

$$\gcd(m,n) = \gcd(|m|,|n|)$$

and that

$$\gcd(m,n) \geq 0$$

# Simple parameter operations

- Thus, we add an absolute-value function

```
#include <iostream>

// Function declarations
int main();
int gcd( int m, int n );
int lcm( int m, int n );
int abs( int n );

// Function definitions
//   ...other function definitions go here...

int abs( int n ) {
    if ( n >= 0 ) {
        return n;
    } else {
        return -n;
    }
}
```

# Simple parameter operations

- We can now update our implementations:

```
int gcd( int m, int n ) {
    m = abs( m );
    n = abs( n );

    assert( (m >= 0) && (n >= 0) );

    return 0;
}

int lcm( int m, int n ) {
    m = abs( m );
    n = abs( n );

    assert( (m >= 0) && (n >= 0) );

    return 0;
}
```

# Easy first, or hard first?

- Next, focus on authoring one function at a time:
  - If there is an easy one, do it first:

```
int lcm( int m, int n ) {
    m = abs( m );
    n = abs( n );

    assert( (m >= 0) && (n >= 0) );

    return (m*n)/gcd( m, n );
}
```

# Default return values

- Other skeletons depend on what is being returned:

```
void function_name_1( … ) {
    return;
}

double function_name_2( … ) {
    return 0.0;
}

char function_name_3( … ) {
    return '\0';
}

bool function_name_4( … ) {
    return false;
}

std::string function_name_5( … ) {
    return "";
}
```

# Summary

- Following this lesson, you now:
  - Know to start with a code skeleton that compiles
  - Check the parameters
    - Make sure it compiles
  - Consider adding helper functions
    - Make sure it compiles
  - Consider if it is appropriate to start with the easiest functions first
    - Make sure it compiles—even if it doesn't return the right answer

# References

[1] Wikipedia: https://en.wikipedia.org/wiki/Assert.h

[2] Cplusplus.com
http://www.cplusplus.com/reference/cassert/

# Acknowledgments

None so far.

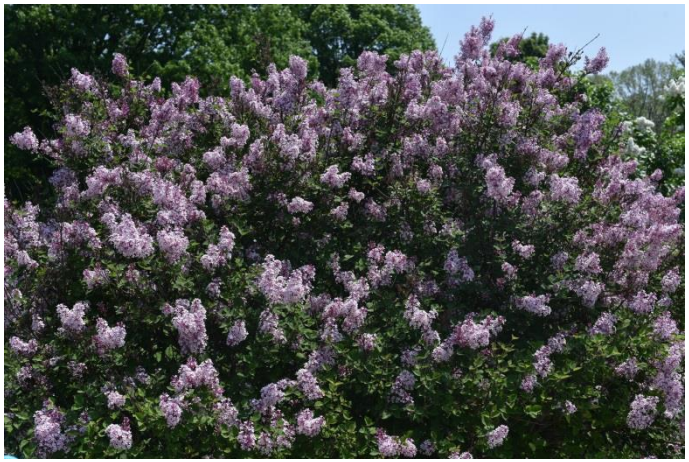
# Colophon

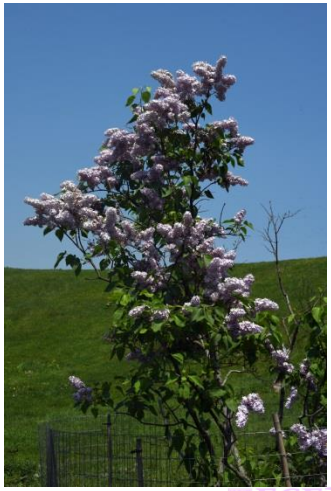
These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

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.