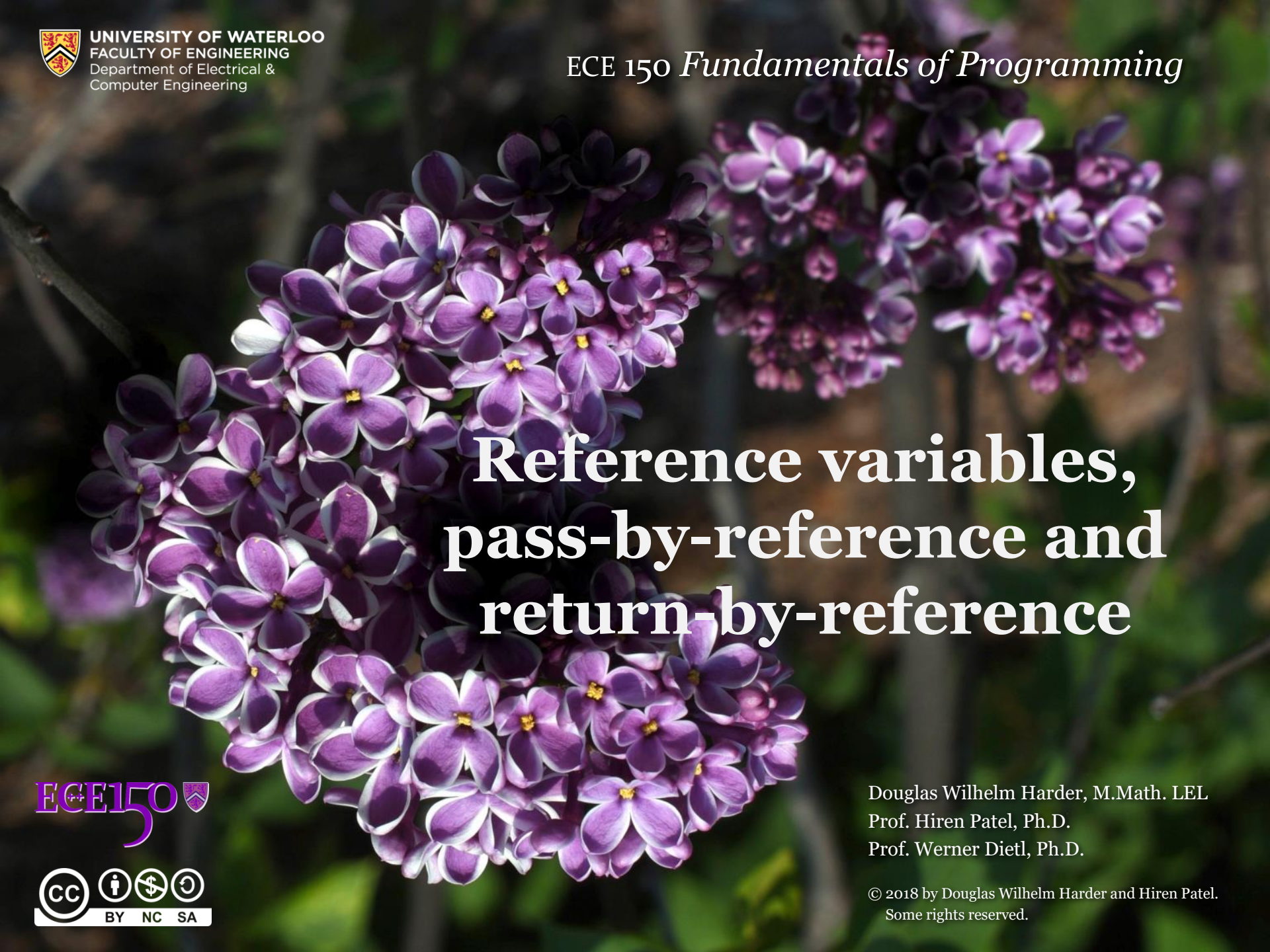

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

Reference variables, pass-by-reference and return-by-reference

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

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

Outline

- In this lesson, we will:
  - Learn about reference variables
    - Aliases to other assignable items (*lvalues*)
  - See how to use this for *pass-by-reference*
    - Changing arguments—not parameters—inside of functions
  - Useful for updating arguments that hold values
  - We will also see *return-by-reference*

CC BY NC SA

Definition

- An *alias* is another name for a person or something
  - Sometimes written *a.k.a.* for also-known-as
  - Mark Twain is an alias for Samuel Langhorne Clemens
  - Charles Lutwidge Dodgson, a.k.a. Lewis Carroll, was a mathematician
- An alias in a programming language is one identifier that is another name for a different identifier

cc BY NC SA

Reference local variables

- Aliases in C++ are through *references variables*

```
typename &new_identifier{ existing_identifier };
```

  - Reference variables must be initialized
  - Whatever they are initialized to must be *assignable*
    - It must be able to be the left-hand side of an assignment operator
    - Anything that can be assigned to is also called an *lvalue*
- Whenever the reference variable is read,
  what lvalue it was initialized with is read
- Whenever the reference variable is assigned to,
  whatever lvalue it was initialized with is assigned to

- An alias does not create a new local variable, parameter, etc.
  - It simply gives another name for an existing identifier

CC BY NC SA

Reference local variables

- For example:

```
#include <iostream>
int main();

int main() {
    int  m{42};
    int &n{m};
    // Now, 'n' is an alias for 'm'
    std::cout << "m = " << m << ", \tn = " << n << std::endl;
    m = 91;
    std::cout << "m = " << m << ", \tn = " << n << std::endl;
    n = 360;
    std::cout << "m = " << m << ", \tn = " << n << std::endl;

    return 0;
}
```

Output:

```
m = 42,     n = 42
m = 91,     n = 91
m = 360,    n = 360
```

cc BY NC SA

Reference local variables

- You could use this to simplify the appearance of your code

```
#include <iostream>
#include <cmath>

int main();

int main() {
    double const &pi{ M_PI };

    // From here on in, you can just use 'pi' instead of 'M_PI'

    return 0;
}
```

`M_PI` is declared `const`,
so your reference must also be `const`

This does not introduce a new local variable

CC BY NC SA

Pass-by-value

- Notice that whenever we called a function,
  the value of the argument was assigned to the parameter
  - This leaves the argument unchanged

```
// Function definitions
void f( int k ) {
   k++;
   std::cout << k << std::endl;
}

int main() {
    int n{42};
    f( 42 );
    f( n );
    f( n + 107 );
    std::cout << "n = " << n << std::endl;

    return 0;
}
```

Output:

```
43
43
150
n = 42
```

cc BY NC SA

Pass-by-reference

- If a parameter is prefixed by an &,
  the parameter is now an alias for the argument
  - Now arguments are restricted to what can be assigned to
    - That is, "*lvalues*"
  - Any change to the parameter changes the value of the argument

CC BY NC SA

Pass-by-reference

- Example:

```
void reset( int &n );

void reset( int &n ) {
    n = 0;
}
```

- Any argument is passed by reference
  - A change to the parameter n also changes the argument

```
int main() {
    int k{ 42 };
    reset( k );
    std::cout << k << std::endl;
    return 0;
}
```

Output:

```
0
```

CC BY NC SA

Pass-by-reference

- Only those items that can appear on the left-hand side of assignment statements can be passed by reference:

```
int main() {
    int k{42};
    reset( k + 1 );
    std::cout << k << std::endl;
    return 0;
}
```

```
example.cpp: In function 'int main()':
example.cpp:12:11: error: invalid initialization of non-const reference of type
'int&' from an rvalue of type 'int'
  reset( k + 1 );
           ^
example.cpp:6:6: error: in passing argument 1 of 'void reset(int&)'
 void reset( int &n ) {
      ^
```

cc BY NC SA

Pass-by-reference

- When you perform a `std::cin` statement,
  the second operand is passed by reference

```
int main() {
    int k;
    std::cout << "Enter an integer: " << std::endl;
    std::cin >> k;
    std::cout << k << "*" << k << " = " << (k*k)
              << std::endl;
    return 0;
}
```

cc BY NC SA

Application: multiple return values

- Suppose you need both the minimum and maximum of three values:

```
void min_max( int a, int b, int c, int &min, int &max ) {
    if ( a < b ) {
        min = a;
        max = b;
    } else {
        min = b;
        max = a;
    }

    if ( c < min ) {
        min = c;
    } else if ( c > max ) {
        max = c;
    }
}
```

cc BY NC SA

Counting time

- Suppose we want to track and print time:
  - You'd need three local variables storing
    - Hours
    - Minutes
    - Seconds
  - Each time a second reaches 60,
    it must reset to 0 and increment the minutes
  - Each time the minutes reaches 60,
    it must reset to 0 and increment the hours
  - Each time the hours reaches 13,
    it must reset to 1,
      but we increment the periods when we reach 12
  - Two periods makes one day

```
10:57:43
10:57:44
10:57:45
10:57:46
10:57:47
10:57:48
10:57:49
10:57:50
10:57:51
10:57:52
10:57:53
10:57:54
10:57:55
10:57:56
10:57:57
10:57:58
10:57:59
10:58:00
   ⋮
 1:44:18
 1:44:19
 1:44:20
 1:44:21
 1:44:22
```

CC BY NC SA

Counting time

```
int main() {
    int hour{10};
    int minute{57};
    int second{42};

    for ( int k{0}; hour < 10000; ++k ) {
        ++second;

        if ( second == 60 ) {
            second = 0;
            ++minute;

            if ( minute == 60 ) {
                minute = 0;
                ++hour;

                if ( hour == 13 ) {
                    hour = 1;
                }
            }
        }
```

cc BY NC SA

Counting time

```
        if ( hour < 10 ) {
            std::cout << " ";
        }

        std::cout << hour << ":";

        if ( minute < 10 ) {
            std::cout << "0";
        }

        std::cout << minute << ":";

        if ( second < 10 ) {
            std::cout << "0";
        }

        std::cout << second << std::endl;

    }

    return 0;
}
```

cc BY NC SA

Counting time

- Suppose you want to increment a variable that stores minutes or seconds:

```
bool increment_minute_second( int &min_sec ) {
    if ( min_sec == 59 ) {
        min_sec = 0;
        return true;
    } else {
        ++min_sec;
        return false;
    }
}
```

cc BY NC SA

Counting time

- Suppose you want to increment a variable that stores hours:

```
bool increment_hour( int &hour ) {
    if ( hour == 12 ) {
        hour = 1;
        return false;
    } else {
        ++hour;
        // Return 'true' if we reach 12 o'clock
        return (hour == 12);
    }
}
```

CC BY NC SA

Counting time

```
 3:30:00
 4:01:57
12:00:00
```

- While we're at it, let's print time nicely
  - Hours may be prefixed by a space or " "
  - Minutes and seconds may be prefixed by a "0"

```
std::string to_string( std::string prefix, int time ) {
    if ( time < 10 ) {
        return prefix + std::to_string( time );
    } else {
        return std::to_string( time );
    }
}
```

These convert an `int` into a `std::string`.

Adding two `std::string` concatenates them.

cc BY NC SA

Counting time

- We can now use this to count time:

```
int main() {
    // Count hours, minutes and seconds starting at 10:57:42
    // breaking at 1:00

    int hour{10};
    int minute{57};
    int second{42};
```

CC BY NC SA

Counting time

```
    for ( int k{0}; k < 10000; ++k ) {
        bool minute_passed{ increment_minute_second( second ) };

        if ( minute_passed ) {
            bool hour_passed{ increment_minute_second( minute ) };

            if ( hour_passed ) {
                increment_hour( hour );
            }
        }

        std::cout << to_string( " ", hour ) << ":"
                  << to_string( "0", minute ) << ":"
                  << to_string( "0", second ) << std::endl;
    }

    return 0;
}
```

BY NC SA

In this course…

- In this course, we will only use pass-by-reference
  - We generally will not use reference variables
  - It is possible to return-by-reference, but that is for another course

cc BY NC SA

Summary

- Following this lesson, you now
  - Know how to create an *alias* or *reference* to another assignable variable
  - Understand that a parameter can be an alias to the argument
    - This is know as *pass-by-reference*
  - Are aware of numerous applications of pass-by-reference
    - Returning more information than one return value allows
  - Know that there is also a *return-by-reference*

cc BY NC SA

References

[1] No references?

CC BY NC SA

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.

CC BY NC SA

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.

CC BY NC SA