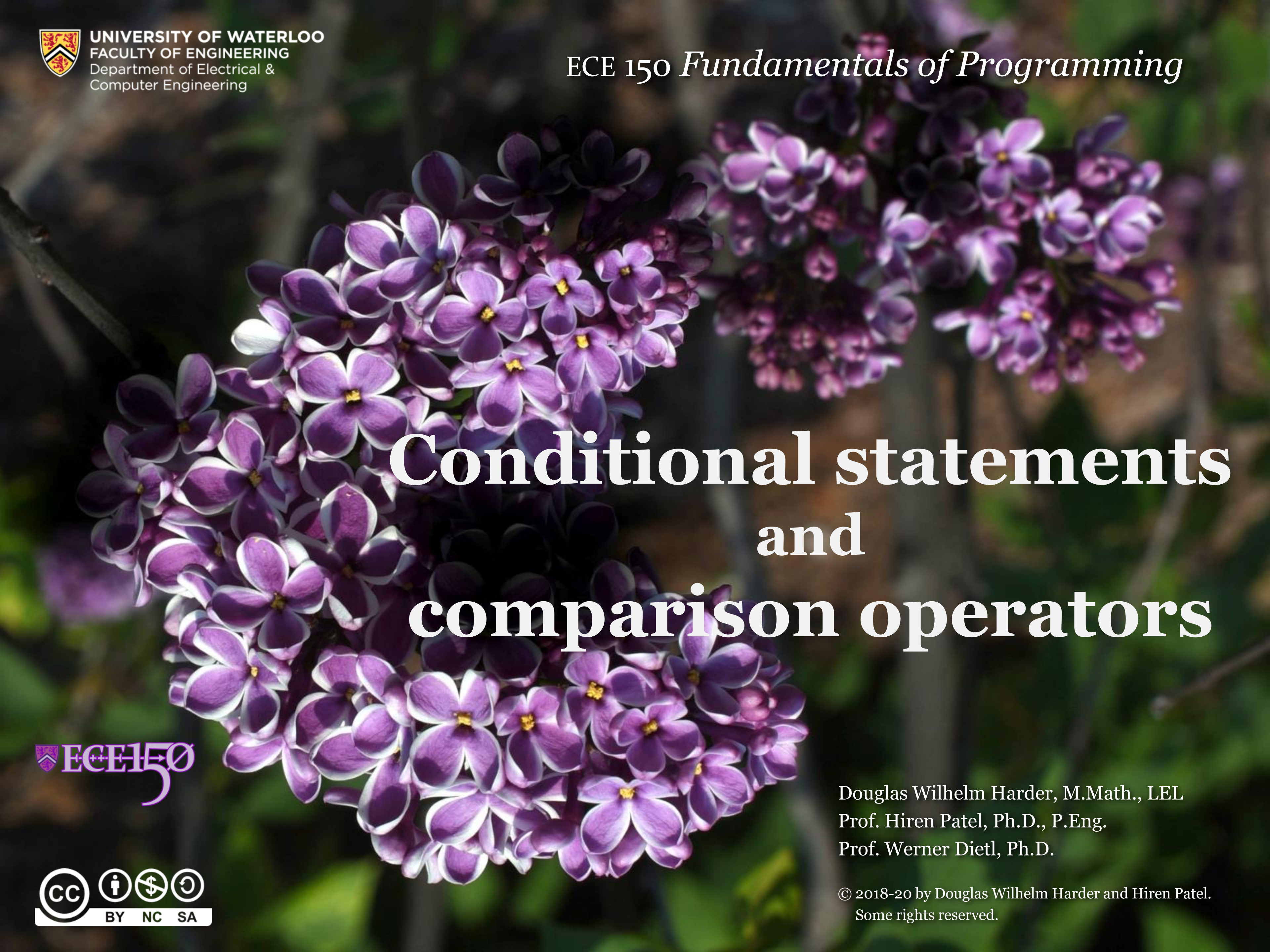
UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*



Conditional statements and comparison operators

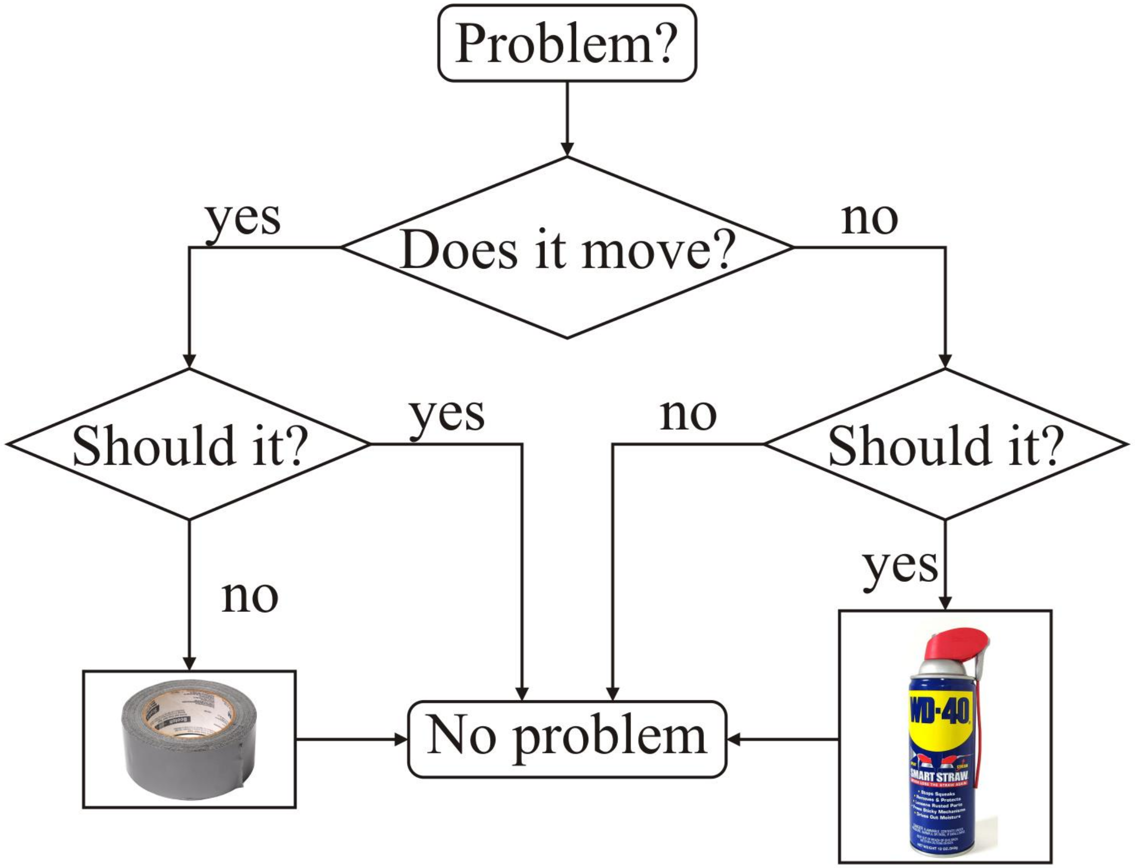
Douglas Wilhelm Harder, M.Math., LEL
Prof. Hiren Patel, Ph.D., P.Eng.
Prof. Werner Dietl, Ph.D.

Outline

- In this lesson, we will:
  - Describe the need for executing code conditionally
  - Describe the flow chart and emphasize the purpose of flow charts
  - Describe the conditional statement
    - The absolute-value and max functions
  - Look at multiple conditional statements
    - Clipping and the tent function
  - Look at a simplification if there is no code to run if the statement is false
    - The sinc function
  - Finally, concluding with a simulation of the operational amplifier

Conditional statements

- In programming, we can *conditionally* execute code if some condition—a Boolean-valued statement—is satisfied (i.e., `true`)



Conditional statements

- Up to this point, we have focused on examples with *serial* execution
  - Each statement in the program is executed one statement at time

- Suppose we only want to execute a statement if a condition is true
  - For example, we may ask the user for a value and then execute code based on that value

Conditional statements

- Here is an example:

```
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
    double x{};
    std::cout << "Enter a number 'x': ";
    std::cin >> x;

    if ( x >= 0 ) {
        std::cout << "|x| = " <<   x  << std::endl;
    } else {
        std::cout << "|x| = " << (-x) << std::endl;
    }

    return 0;
}
```

If the user entered a positive number,
the first block of statements is executed;
otherwise, the second block of statements is executed.

Conditional statements

- In order to choose which block of code to execute based on a given condition, we use a conditional statement:

```
    if ( Boolean-valued condition ) {
        // The consequent block or body of statements
        //  - to be executed if the condition is true
    } else {
        // The alternative block or body of statements
        //  - to be executed if the condition is false
    }
```

- Even though a conditional statement may have many statements within it, the entire structure is referred to as a conditional statement

Conditional statements

- In order to execute code only if some condition is satisfied, we use a conditional statement:

```
if ( Boolean-valued condition ) {
    // The consequent block or body of statements
    //   - to be executed if the condition is true
}
```

Conditional statements

- A Boolean-valued condition is any test that returns `true` or `false`
- We will look at six such conditions:
  - These are called the *binary comparison operators*
  - Each takes two operands, each returns `true` or `false`

Operator	Example
Less than	`x < y`
Greater than	`x > y`
Less-than or equal to	`x <= y`
Greater-than or equal to	`x >= y`
Equals	`x == y`
Does not equal	`x != y`

Conditional statements

- It is incredibly important to remember that to test equality, you must use the == operator and not the = operator
  - The = operator is the assignment operator

- You must use <= and >=
  - You cannot use =< and =>
  - Write it as you say it:

    Less than or equal to

    < =

- Think of "!" as meaning *not*
  - Thus, the != operator is the *not equals operator*

The max function

- As a second example, the maximum of two values is also based on a simple condition:

$$\max(x, y) \overset{\text{def}}{=} \begin{cases} x & x \geq y \\ y & x < y \end{cases}$$

  - Let’s write a program that prints the maximum of two values:

The max function

$$\max(x, y) \overset{\text{def}}{=} \begin{cases} x & x \geq y \\ y & x < y \end{cases}$$

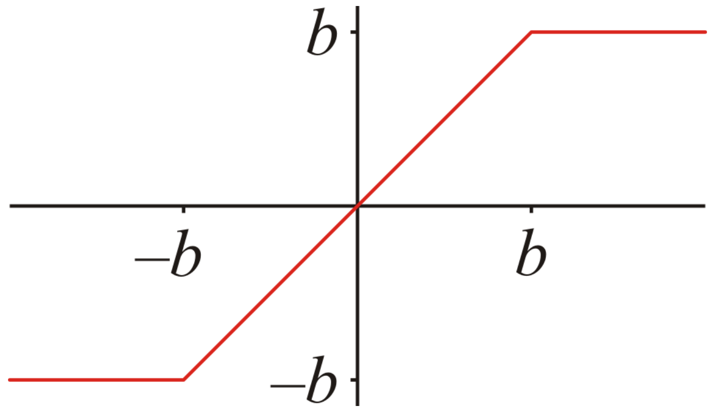
- Here is an implementation:

```
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
    double x{};
    std::cout << "Enter a number 'x': ";
    std::cin >> x;

    double y{};
    std::cout << "Enter a number 'y': ";
    std::cin >> y;

    if ( x >= y ) {
        std::cout << "max(x, y) = " << x << std::endl;
    } else {
        std::cout << "max(x, y) = " << y << std::endl;
    }

    return 0;
}
```

Clipping signals

- In engineering, signals (values) often cannot exceed certain bounds
  - If a signal x is greater in absolute value than some bound b, the bound is returned

$$\text{clip}_b(x) \overset{\text{def}}{=} \begin{cases} b & x \geq b \\ -b & x \leq -b \\ x & \text{otherwise} \end{cases}$$



Clipping signals

- Here is an implementation:

```
// Function definitions
int main() {
    double x{};
    std::cout << "Enter a number 'x': ";
    std::cin >> x;

    double bound{};
    std::cout << "Enter a bound: ";
    std::cin >> bound;

    if ( x >= bound ) {
        std::cout << "clip(x) = " << bound << std::endl;
    } else {
        // x < bound
        if ( x <= -bound ) {
            std::cout << "clip(x) = " << (-bound) << std::endl;
        } else {
            std::cout << "clip(x) = " <<   x      << std::endl;
        }
    }

    return 0;
}
```

$$\text{clip}_b(x) \overset{\text{def}}{=} \begin{cases} b & x \geq b \\ -b & x \leq -b \\ x & \text{otherwise} \end{cases}$$

Clipping signals

- In this example, there are three non-overlapping cases:
  1. When $x \geq b$
  2. When $x \leq -b$
  3. When $-b < x < b$

- We can instead write such a conditional statement as

```
if ( x >= bound ) {
    std::cout << "clip(x) = " <<  bound << std::endl;
} else if ( x <= -bound ) {
    std::cout << "clip(x) = " << -bound << std::endl;
} else {
    // If neither of the two previous conditions
    // is true, then -bound < x < bound
    std::cout << "clip(x) = " <<  x     << std::endl;
}
```

Cascading conditional statements

- Such a sequence of if—else-if—··· statements is referred to as
  cascading *conditional statements*

```
if ( condition-1 ) {
    // First consequent block
    // Do something
} else if ( condition-2 ) {
    // Second consequent block
    // Do something else
} else {
    // Complementary
    //    alternative block
    // Do something else,
    // yet again...
}
```

Martin Püschel, *Song Khon Waterfall*

Cascading conditional statements

- You can have as many conditions as is deemed necessary

```
if ( condition-1 ) {
    // First consequent block
    // Do something
} else if ( condition-2 ) {
    // Second consequent block
    // Do something else
} else if ( condition-3 ) {
    // Third consequent block
    // Do something else
} else {
    // Complementary
    //    alternative block
    // Do something else,
    // yet again...
}
```

Martin Püschel, *Song Khon Waterfall*

Cascading conditional statements

- As before, it is not necessary to have a complementary alternative block

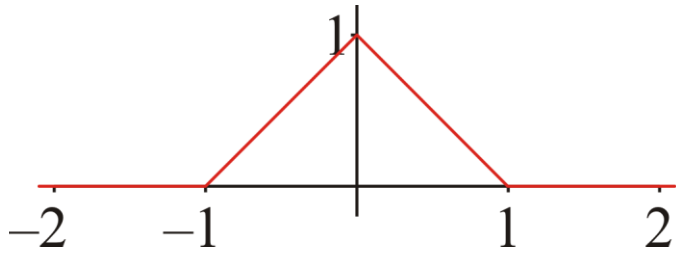
```
if ( condition-1 ) {
    // First consequent block
    // Do something
} else if ( condition-2 ) {
    // Second consequent block
    // Do something else
} else if ( condition-3 ) {
    // Third consequent block
    // Do something else
}
```

Martin Püschel, *Song Khon Waterfall*

The *tent* function

- A *tent function* is defined as:

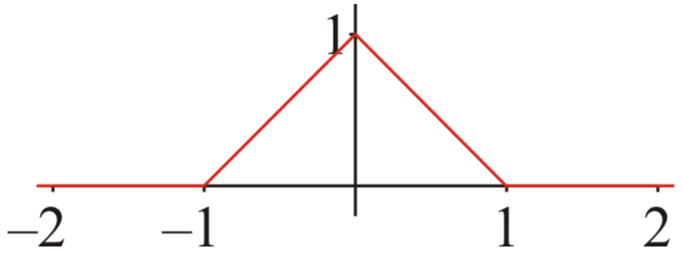
$$\operatorname{tent}(x) \overset{\text{def}}{=} \begin{cases} 0 & x \le -1 \\ x+1 & -1 < x \le 0 \\ 1-x & 0 < x \le 1 \\ 0 & x > 1 \end{cases}$$



Cascading conditional statements

- Here is an implementation:

```
// Function definitions
int main() {
    double x{};
    std::cout << "Enter a number 'x': ";
    std::cin >> x;

    if ( x <= -1.0 ) {
        std::cout << "tent(x) = " <<  0.0      << std::endl;
    } else if ( x <= 0.0 ) {
        std::cout << "tent(x) = " << (x + 1.0) << std::endl;
    } else if ( x <= 1.0 ) {
        std::cout << "tent(x) = " << (1.0 - x) << std::endl;
    } else {
        std::cout << "tent(x) = " <<  0.0      << std::endl;
    }

    return 0;
}
```



Once a condition is checked and evaluates to true, no subsequent conditions are checked

How not to use cascades

- Novice programmers sometimes want to emphasize the conditional checks:

```
if ( x <= 0 ) {
    // Do something...
} else if ( x > 0 ) {
    // Do something else...
}
```

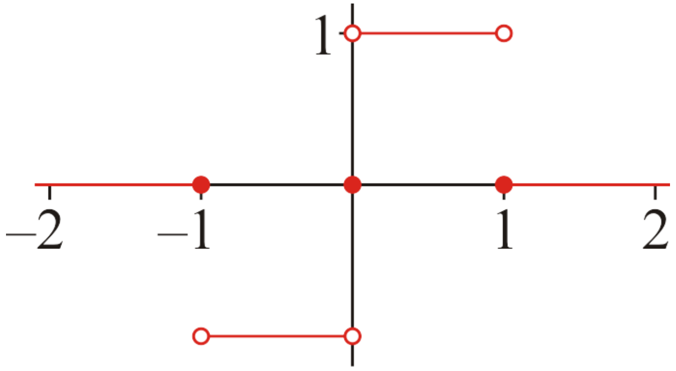
```
if ( x == 0 ) {
    // Do something...
} else if ( x != 0 ) {
    // Do something else...
}
```

- Don’t do this:
  – The second condition is complementary to the first
  – Experienced programmers reading this will be confused
    - They expect that there are some values of x that satisfy neither condition
  – Maintenance becomes more difficult

Common errors with cascades

- Consider this code:

```
if ( x < -1.0 ) {
    std::cout <<  0.0;
} else if ( x < 0.0 ) {
    std::cout << -1.0;
} else if ( x > 0.0 ) {
    std::cout <<  1.0;
} else if ( x > 1.0 ) {
    std::cout <<  0.0;
} else {
    std::cout <<  0.0;
}
```



- What are the errors in this cascade?

Summary

- Following this lesson, you now:
  - Understand the format of a conditional statement:
    - A Boolean-valued condition
    - A consequent block of statements to be executed if the condition is `true`
    - An alternative block of statements to be executed if the condition is `false`
  - Know that the condition may be a comparison:
    - One of six comparison operators with two operands
  - Understand alternative block is not required
  - Know how to have a cascading conditional statement with two or more conditions, each with their own associated block of statements

References

[1] Wikipedia
https://en.wikipedia.org/wiki/Conditional_(computer_programming)

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