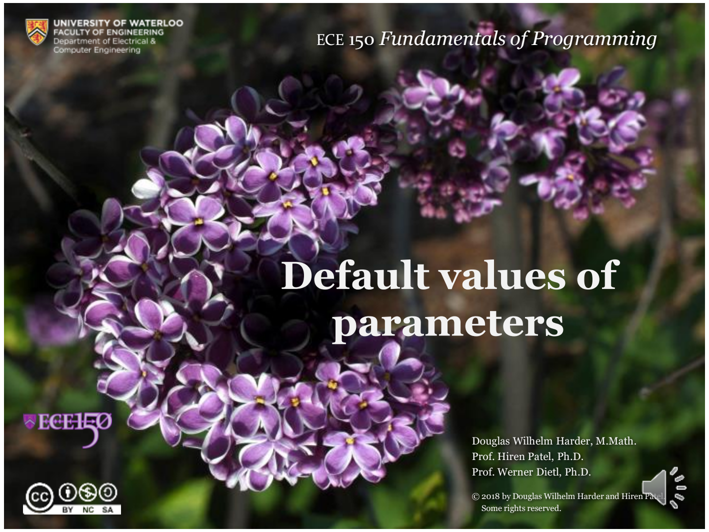
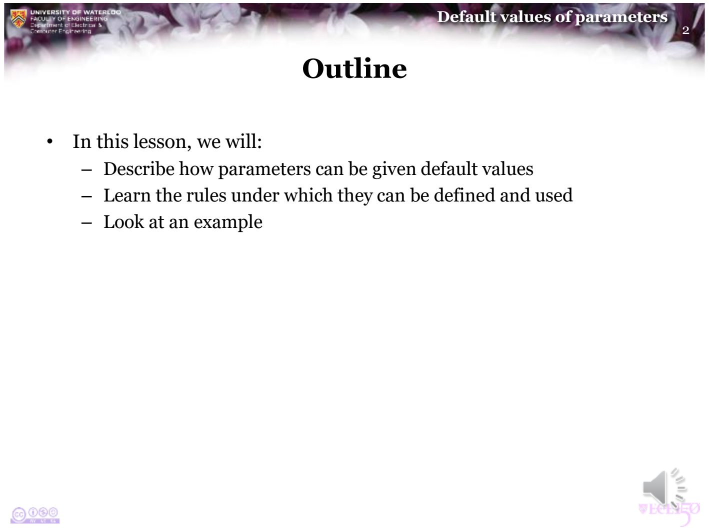


UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 150 *Fundamentals of Programming*

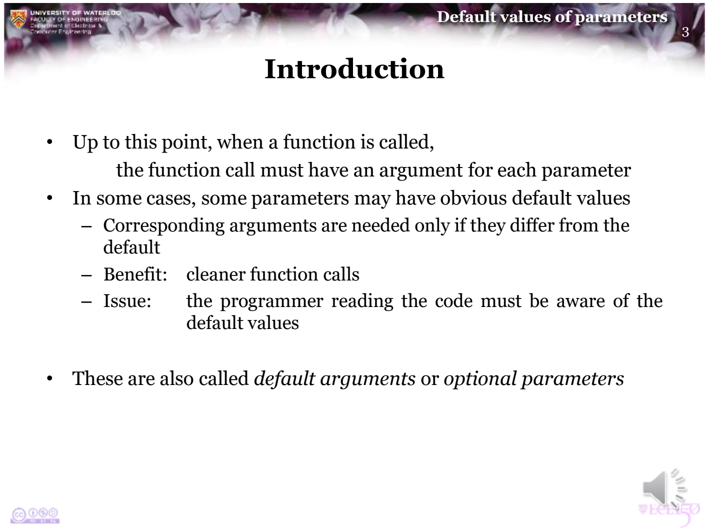
# Default values of parameters

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



## Outline

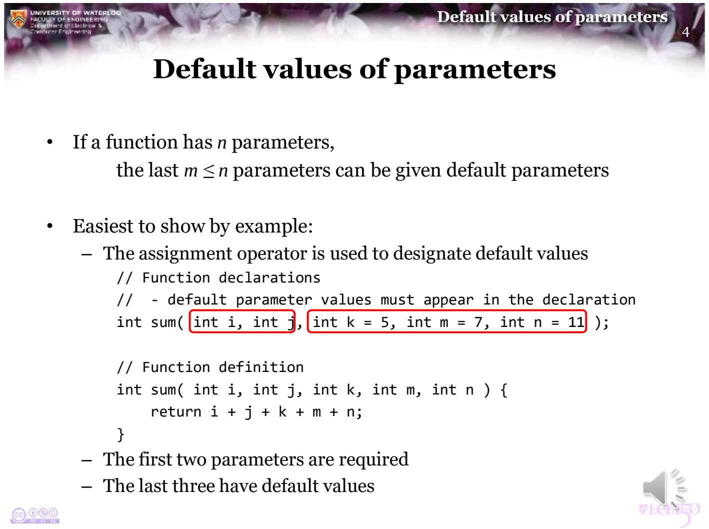
- In this lesson, we will:
  - Describe how parameters can be given default values
  - Learn the rules under which they can be defined and used
  - Look at an example



## Introduction

- Up to this point, when a function is called,
  the function call must have an argument for each parameter
- In some cases, some parameters may have obvious default values
  - Corresponding arguments are needed only if they differ from the default
  - Benefit: cleaner function calls
  - Issue: the programmer reading the code must be aware of the default values

- These are also called *default arguments* or *optional parameters*



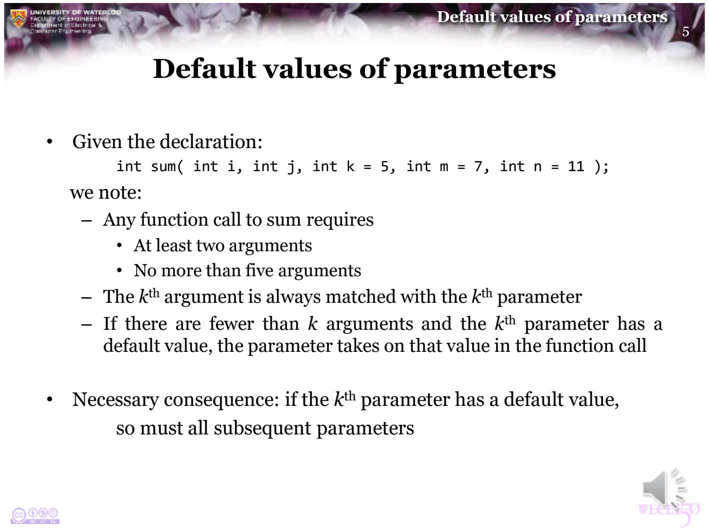
## Default values of parameters

- If a function has *n* parameters,
  the last $m \le n$ parameters can be given default parameters

- Easiest to show by example:
  - The assignment operator is used to designate default values

```
// Function declarations
//  - default parameter values must appear in the declaration
int sum( int i, int j, int k = 5, int m = 7, int n = 11 );

// Function definition
int sum( int i, int j, int k, int m, int n ) {
    return i + j + k + m + n;
}
```

  - The first two parameters are required
  - The last three have default values

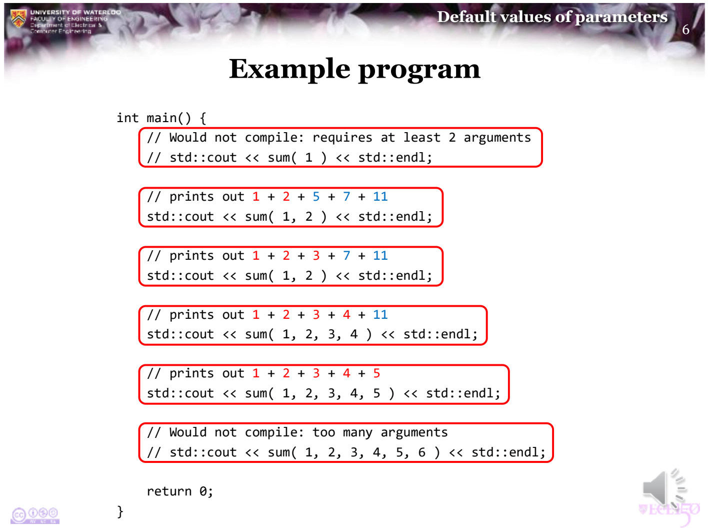


# Default values of parameters

- Given the declaration:

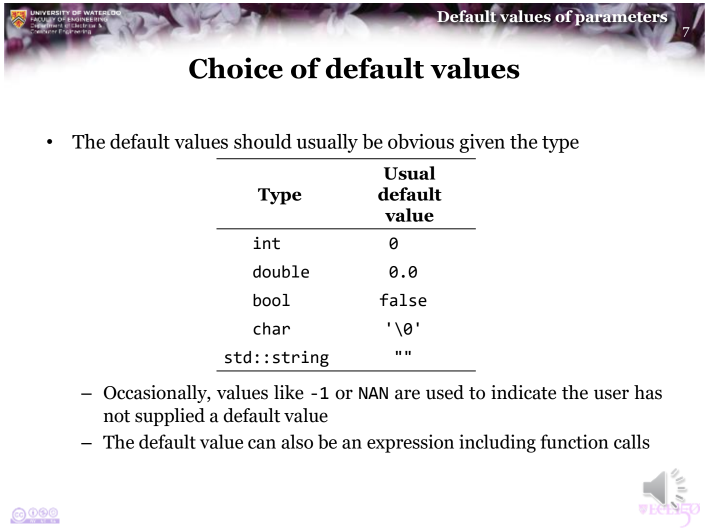
```
int sum( int i, int j, int k = 5, int m = 7, int n = 11 );
```

we note:
  - Any function call to sum requires
    - At least two arguments
    - No more than five arguments
  - The $k^{\text{th}}$ argument is always matched with the $k^{\text{th}}$ parameter
  - If there are fewer than $k$ arguments and the $k^{\text{th}}$ parameter has a default value, the parameter takes on that value in the function call
- Necessary consequence: if the $k^{\text{th}}$ parameter has a default value, so must all subsequent parameters



# Example program

```
int main() {
    // Would not compile: requires at least 2 arguments
    // std::cout << sum( 1 ) << std::endl;

    // prints out 1 + 2 + 5 + 7 + 11
    std::cout << sum( 1, 2 ) << std::endl;

    // prints out 1 + 2 + 3 + 7 + 11
    std::cout << sum( 1, 2 ) << std::endl;

    // prints out 1 + 2 + 3 + 4 + 11
    std::cout << sum( 1, 2, 3, 4 ) << std::endl;

    // prints out 1 + 2 + 3 + 4 + 5
    std::cout << sum( 1, 2, 3, 4, 5 ) << std::endl;

    // Would not compile: too many arguments
    // std::cout << sum( 1, 2, 3, 4, 5, 6 ) << std::endl;

    return 0;
}
```

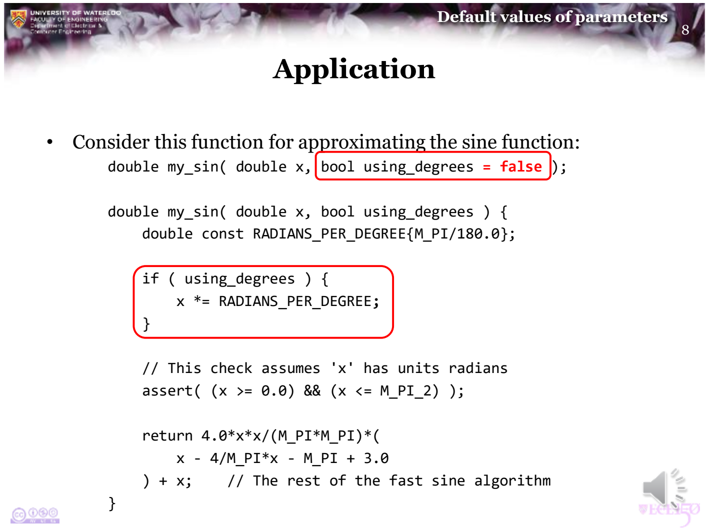


# Choice of default values

- The default values should usually be obvious given the type

| Type | Usual default value |
|---|---|
| `int` | `0` |
| `double` | `0.0` |
| `bool` | `false` |
| `char` | `'\0'` |
| `std::string` | `""` |

  - Occasionally, values like `-1` or `NAN` are used to indicate the user has not supplied a default value
  - The default value can also be an expression including function calls



# Application

- Consider this function for approximating the sine function:

```
double my_sin( double x, bool using_degrees = false );

double my_sin( double x, bool using_degrees ) {
    double const RADIANS_PER_DEGREE{M_PI/180.0};

    if ( using_degrees ) {
        x *= RADIANS_PER_DEGREE;
    }

    // This check assumes 'x' has units radians
    assert( (x >= 0.0) && (x <= M_PI_2) );

    return 4.0*x*x/(M_PI*M_PI)*(
        x - 4/M_PI*x - M_PI + 3.0
    ) + x;    // The rest of the fast sine algorithm
}
```

# Summary

- Following this lesson, you now:
  - Understand the use and application of default values for parameters
  - Know the syntax:
    - All parameters after the first with a default value must have a default value
    - Default values cannot be skipped
    - The default values should be the obvious values

# References

[1] No references?

# 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.