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

ECE 150 *Fundamentals of Programming*

Writing functions

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

Outline

- In this lesson, we will:
  - Describe the idea of reusing code
    - Specifically, the issues with cutting and pasting code
  - Describe functions in C++
  - Look at how parameters can be used in the function body
  - We will look at implementing
    - A simple sine function
    - An absolute value function
    - The maximum of two parameters
    - A function that evaluates a cubic polynomial at a point
    - The maximum of three parameters
  - Look at how to use a function call within a function
  - See numerous examples

Using other code

- To this point, we have described:
  - Function declarations, including

```
int main();
double cos( double x );
double pow( double x, double y );
```

  - One function definition:

```
int main() {
    // Do something, if you want...
    return 0;
}
```

Review of functions

- Suppose we are authoring a piece of code that requires us to repeatedly calculate the absolute value of a local variable:

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

    double y{3.14159265358979323846 - x };

    if ( y < 0 ) {
        y = -y;
    }

    std::cout << "|x - pi| = " << y << std::endl;

    return 0;
}
```

Review of functions

- Suppose we are authoring a piece of code that requires us repeatedly determine the maximum of two values:

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

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

    // Set z = max( x, y );
    double z{x};

    if ( y > z ) {
        z = y;
    }

    std::cout << "max(x, y) = " << z << std::endl;

    return 0;
}
```

Functions in C++

- First, note that it is not immediately obvious what each of these code fragments are doing
  - You must examine the code, and deduce what is expected
    - Either that, or perhaps read the comments
  - You have to repeat this every time you want to perform these operations
    - What is the likelihood that each time,
      you will not make a mistake (introduce a bug)

Functions in C++

- If you find yourself performing a similar option more than once,
  you are essentially cutting-and-pasting code
- Questions:
  - What happens if you update one and forget the others?
  - What happens if you find a bug in one but forget you copied that code elsewhere before?

Functions in C++

- A function is a means of authoring a piece of code to perform one operation once
  - Once it is tested, you and anyone else can repeatedly use it without ever having to worry about it
  - We have already seen numerous functions implemented in the C math library
    - You don't care how they are implemented
    - Similarly, people won't care how your functions are implemented

A simple sine function

- Here is a function declaration for my sine function:

```
double my_sin( double x );
```

- The function definition has a body that is executed when the function is called:

function identifier or function name

return type

parameter and parameter type

```
double my_sin( double x ) {
    double result{ x - x*x*x/6.0 + x*x*x*x*x/120.0 };
    return result;
}
```

local variable

function body

the returned value

A simple sine function

```
double my_sin( double x ) {
    double result{ x - x*x*x/6.0
                     + x*x*x*x*x/120.0 };
    return result;
}
```

- We can now use this:

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <iostream>

// Function declarations
int main();
double my_sin( double x );

// Function definitions
int main() {
    std::cout <<   my_sin( 0.4 ) << std::endl;
    std::cout << std::sin( 0.4 ) << std::endl;

    return 0;
}

// Insert function definition of my_sin here
```

argument

Output:

```
0.389419
0.389418
```

An absolute value function

- Here is the declaration of the absolute value function:

```
double my_abs( double x );
```

- The function definition has a body that is executed when the function is called:

```
double my_abs( double x ) {
    double result{};
    // Do something to calculate |x|
    return result;
}
```

An absolute value function

- For this function

```
double my_abs( double x );
```

- The identifier x is called a *parameter* of the function
  - For different arguments, the output of the function may be different
    - That is, the identifier *parameterizes* the function
- The scope of the parameter is restricted to the function body
  - Like a local variable, it can be accessed and modified
  - Its initial value is the value of the argument passed to the function in the corresponding location during the function call

An absolute value function

- What is the expected return value?
  - If the parameter `x >= 0`,
    the function should return `x`
  - If the parameter `x < 0`,
    the function should return `-x`
- Thus, one implementation of the function body is:

```
double my_abs( double x ) {
    double result{};

    if ( x >= 0 ) {
        result = x;
    } else {
        result = -x;
    }

    return result;
}
```

An absolute value function

- We can now use this function:

```
int main() {
    std::cout << my_abs(  -0.3 ) << std::endl;
    std::cout << my_abs(   0.1 ) << std::endl;
    std::cout << my_abs( -15.9 ) << std::endl;

    return 0;
}
```

A function returning a polynomial

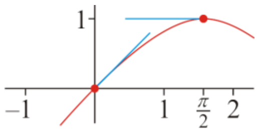
- Suppose we have a cubic spline:

$$p(x) \overset{\text{def}}{=} \boxed{\frac{4x^2}{\pi^2}}\left(\boxed{x - \frac{4}{\pi}x - \pi + 3}\right)\boxed{+x}$$

```
#define _USE_MATH_DEFINES
#include <cmath>

// Function declarations
double spline( double x );
int main();

// Function definitions
double spline( double x ) {
    return 4.0*x*x/(M_PI*M_PI)*(
        x - 4/M_PI*x - M_PI + 3.0
    ) + x;
}
```



A function returning a polynomial

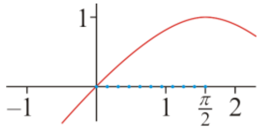
- Inside `main()`, we can use this function:

```
int main() {
    for ( int k{0}; k <= 10; ++k ) {
        std::cout << M_PI_2*(0.1*k) << ",\t"
                  << spline( M_PI_2*(0.1*k) ) << ",\t"
                  << std::sin( M_PI_2*(0.1*k) ) << std::endl;
    }

    return 0;
}
```

Output:

```
0,        0,        0
0.15708,            0.155235,           0.156434
0.314159,           0.305062,           0.309017
0.471239,           0.446907,           0.45399
0.628319,           0.578195,           0.587785
0.785398,           0.69635,            0.707107
0.942478,           0.798796,           0.809017
1.09956,            0.88296,            0.891007
1.25664,            0.946265,           0.951057
1.41372,            0.986137,           0.987688
1.5708, 1,          1
```



A max function

- Here is a different use of return:

```
// Function declarations
double max( double x, double y );

// Function definitions
double max( double x, double y ) {
    if ( x >= y ) {
        return x;
    } else {
        return y;
    }
}
```

Functions used in algebraic expressions

- Here we see a function call inside an argument to a function call:

```
#include <iostream>

// Function declarations
double my_abs( double x );
int main();

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

    std::cout << my_abs( my_abs( z ) - 1.0 ) << std::endl;
    return 0;
}
```

Functions used in algebraic expressions

- We could use the absolute value function inside another function:

```
// Function declarations
double my_abs( double x );
double W( double x );
int main();

// Function definitions
double W( double x ) {
    return my_abs( my_abs( 3.0*x ) - 3.0 );
}
```

$$\left||3x|-3\right|$$

Functions used in algebraic expressions

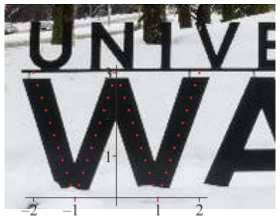
- Here we use the function W in a program:

```
#include <iostream>

// Function declarations
double abs( double x );
double W( double x );
int main();

// Function definitions
int main() {
    for ( int k{-20}; k <= 20; ++k ) {
        std::cout << 0.1*k << ", " << W( 0.1*k ) << std::endl;
    }
}

// Include function definitions for my_abs and W
```



The maximum of three parameters

- As the requirements for the function become more complex, so do the implementations

```
// Function declaration
double max( double x, double y, double z );

// Function definition
double max( double x, double y, double z ) {
    if ( (x >= y) && (x >= z) ) {
        return x;
    } else if ( y >= z ) {
        return y;
    } else {
        return z;
    }
}
```

The maximum of three parameters

- Although, at times, perhaps, other functions can be used to simplify the implementation

```
// Function declaration
double max( double x, double y, double z );

// Function definition
double max( double x, double y, double z ) {
    return max( max( x, y ), z );
}
```

A factorial function

- Anything you have authored in `main()` can also be part of a function
  - Consider this implementation of the factorial

```
// Function declaration
int factorial( int n );

// Function definition
int factorial( int n ) {
    int result{1};

    for ( int k{1}; k <= n; ++k ) {
        result *= k;
    }

    return result;
}
```

$$n! = n\cdot(n-1)\cdot(n-2)\cdots 3\cdot 2\cdot 1$$
$$= 1\cdot 2\cdot 3\cdots(n-2)\cdot(n-1)\cdot n$$

Variations on the absolute value function

- There are many equally valid ways of writing the same function:

```
double abs( double x ) {
    if ( x >= 0 ) {
        return x;
    } else {
        return -x;
    }
}
```

```
double abs( double x ) {
    double result{x};

    if ( result < 0 ) {
        result = -result;
    }

    return result;
}
```

```
double abs( double x ) {
    double result{};

    if ( x >= 0 ) {
        result = x;
    } else {
        result = -x;
    }

    return result;
}
```

```
double abs( double x ) {
    if ( x < 0 ) {
        x = -x;
    }

    return x;
}
```

Variations on the maximum of two parameters

- There are many equally valid ways of writing the same function:

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

```
double max( double x , double y ) {
    double result{x};

    if ( result < y ) {
        result = y;
    }

    return result;
}
```

```
double max( double x , double y ) {
    if ( x < y ) {
        x = y;
    }

    return x;
}
```

Variations on the maximum of three parameters

- There are many equally valid ways of writing the same function:

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

```
double max( double x , double y , double z ) {
    if ( x < y ) {
        x = y;
    }

    if ( x < z ) {
        x = z;
    }

    return x;
}
```

```
double max( double x , double y , double z ) {
    double result{x};

    if ( result < y ) {
        result = y;
    }

    if ( result < z ) {
        result = z;
    }

    return result;
}
```

Variations on the maximum of three parameters

- There are many equally valid ways of writing the same function:

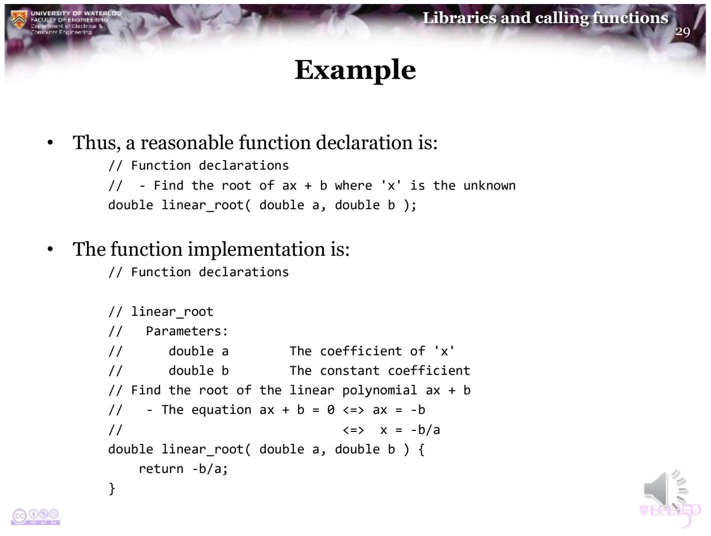
```
double max( double x, double y, double z ) {
    if ( x >= y ) {
        if ( x >= z ) {
            return x;
        } else {
            return z;
        }
    } else {
        if ( y >= z ) {
            return y;
        } else {
            return z;
        }
    }
}
```

Example

- Let's write a function that solves a problem:
  - Let us write a function that returns the root of $ax + b$
  - Now, if $ax + b = 0$

$$ax = -b$$
$$x = -\frac{b}{a}$$

  - Note that we don't care about the variable:
    - $-\frac{b}{a}$ is the root of $ay + b$ and $a\xi + b$
  - Thus, the only information we need are the values of the coefficients



Example

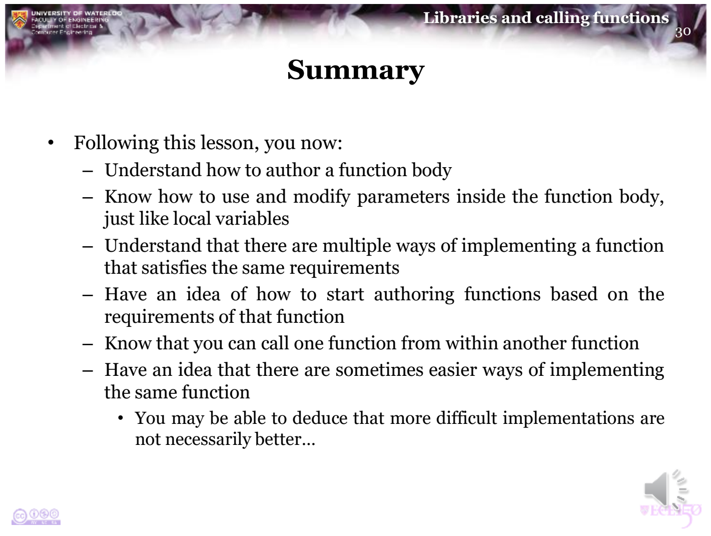
- Thus, a reasonable function declaration is:

```
// Function declarations
//  - Find the root of ax + b where 'x' is the unknown
double linear_root( double a, double b );
```

- The function implementation is:

```
// Function declarations

// linear_root
//   Parameters:
//      double a        The coefficient of 'x'
//      double b        The constant coefficient
// Find the root of the linear polynomial ax + b
//   - The equation ax + b = 0 <=> ax = -b
//                             <=>  x = -b/a
double linear_root( double a, double b ) {
    return -b/a;
}
```

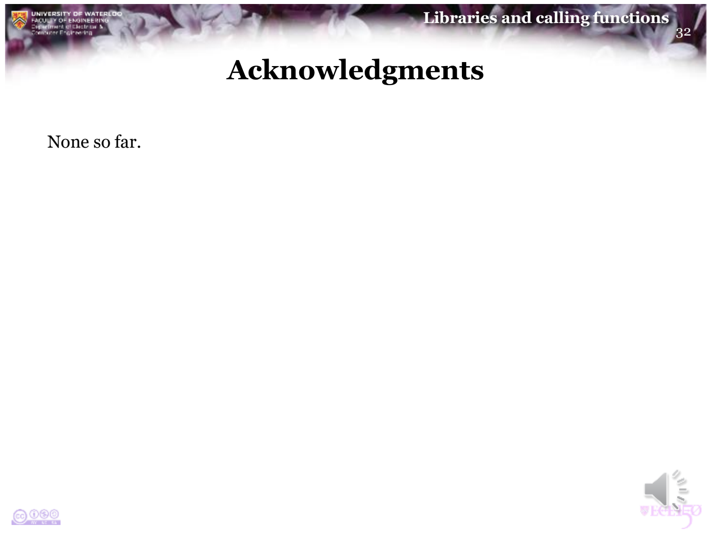


Summary

- Following this lesson, you now:
  - Understand how to author a function body
  - Know how to use and modify parameters inside the function body, just like local variables
  - Understand that there are multiple ways of implementing a function that satisfies the same requirements
  - Have an idea of how to start authoring functions based on the requirements of that function
  - Know that you can call one function from within another function
  - Have an idea that there are sometimes easier ways of implementing the same function
    - You may be able to deduce that more difficult implementations are not necessarily better…

References

[1] Cplusplus.com
http://www.cplusplus.com/reference/cmath/



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