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

ECE 150 *Fundamentals of Programming*

Libraries and calling functions

ECE150

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

Outline

- In this lesson, we will:
  - Describe the concept of a library
  - Look at the C math library
  - Look at the global variables defined in that library
  - Describe a function declaration
  - Look at how to call a function with arguments
  - Understand how to use the returned value
  - Look at a number of examples

Using other code

- In our examples so far, we have performed
  - Arithmetic, comparison and logical operations
  - Assignments
  - Conditional statements and repetition statements
- How do we use someone else's code?
  - Suppose we are simulating a cyclic system and require the use of trigonometric functions?

Review of functions

- Recall from secondary school that functions took arguments:

  $\sin(x)$ $\ln(x)$ $\gcd(m, n)$

  - For example, you may have used

    $\sin(\pi/6)$ $\ln(3)$ $\gcd(15, 18)$

- In secondary school, you may have said:
  - The sine function takes a real value as an argument and returns a real value on the interval $[-1, 1]$
  - The natural logarithm takes a positive real value as an argument and returns a real value
  - The GCD takes two integers and returns a positive integer

Functions in C++

- A function in C++ is a body of instructions that:
  - Allows you to specify certain parameters
  - Calculates or *returns* a value based on those parameters
- For example, there are C++ functions that calculate:
  sin(*x*) ln(x) gcd(*m*, *n*)
- Functions avoid the need to *reinvent the wheel*
  - E.g., someone else has already authored the trigonometric functions

Libraries

- The solution for collecting related functions together for re-use in C++ is a *library*
  - Suppose you need information on something:
    - You go to an appropriate library, and access that book
  - A C++ library can be
    - A collection of functions that you can call from your program
    - A collection of global variables you can access
    - Other objects and classes associated with object-oriented design
      - We will see this later in this course

Libraries

- We will examine the C math library
  - This is a collection of mathematical constants and functions
- You access the C math library by including
  `#include <cmath>`
- As you may guess, `iostream` is another library
  `#include <cmath>`

Global variables

- First, the global variables, all of type `double`:

`M_E`	e	2.71828182845904523536
`M_LOG2E`	$\log_2(e)$	1.44269504088896340736
`M_LOG10E`	$\log_{10}(e)$	0.434294481903251827651
`M_LN2`	ln(2)	0.693147180559945309417
`M_LN10`	ln(10)	2.30258509299404568402
`M_PI`	π	3.14159265358979323846
`M_PI_2`	$\pi/2$	1.57079632679489661923
`M_PI_4`	$\pi/4$	0.785398163397448309616
`M_1_PI`	$1/\pi$	0.318309886183790671538
`M_2_PI`	$2/\pi$	0.636619772367581343076
`M_2_SQRTPI`	$2/\sqrt{\pi}$	1.12837916709551257390
`M_SQRT2`	$\sqrt{2}$	1.41421356237309504880
`M_SQRT1_2`	$1/\sqrt{2}$	0.707106781186547524401

Example

- These can be used in a program:

```
#define _USE_MATH_DEFINES    // This is required for some IDEs and compilers
#include <cmath>
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
    double radius{};
    std::cout << "Enter the radius of a sphere: ";
    std::cin >> radius;

    double area{4.0*M_PI*radius*radius};
    double volume{(4.0/3.0)*M_PI*radius*radius*radius};

    std::cout << "The surface area is " << result << std::endl;
    std::cout << "The volume is "       << volume << std::endl;

    return 0;
}
```

Output:

```
Enter the radius of a sphere: 2

The surface area is 50.2655
The volume is 33.5103
```


Function declarations

- The C math library also contains a number of *functions* you can use
  - Each function is described by the *function declaration*
  - The function declaration for the sine function is

```
double sin( double x );
```

It evaluates to or *returns* a float — It takes a single float as an *argument*

  - The function that calculates x^y is

```
double pow( double x, double y );
```

It evaluates to or *returns* a float — It takes two floats as arguments

Function declarations

- If this library had a gcd function, its function declaration would be:

```
int gcd( int m, int n );
```

It evaluates to or *returns* an integer — It takes two integers as arguments

- The function declaration may also be described as the:
  - signature
  - prototype
  - interface

  of the function

- You have already seen one function declaration in this course:

```
int main();
```


Trigonometric functions

- Some function declarations in the `cmath` library are:

```
double cos( double x );
double sin( double x );
double tan( double x );
double acos( double x );
double asin( double x );
double atan( double x );
double atan2( double y, double x );
```

- Immediately, you may notice:
  - If you want sec(x), you must use 1/cos(x)
  - More interesting is `atan2(…)`:
    - It returns $\tan^{-1}(y/x)$, but takes into account the sign of y and x
    - This allows, for example, $x = 0$

Trigonometric functions

- Here are examples of *calling functions*
  - That is, we are *calling* the function with an argument or arguments

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

// Function declarations
int main();

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

    std::cout << " sin(" << x << ") = " << std::sin(x) << std::endl;
    std::cout << " cos(" << x << ") = " << std::cos(x) << std::endl;
    std::cout << "atan(" << x << ") = " << std::atan(x) << std::endl;

    return 0;
}
```

Output:

```
Enter a real value 'x': 3.14

 sin(3.14) = 0.00159265
 cos(3.14) = -0.999999
atan(3.14) = 1.26248
```


Functions calls in arithmetic expressions

- In any arithmetic expression where you could have used a float or local variable of type float, you can also use a call to a function that has a return type double

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

    double y{std::sin(x) + 1.0};
    std::cout << "sin(" << x << ") + 1 = " << y << std::endl;

    y = std::sin(std::asin(x));
    std::cout << "sin(asin(" << x << ")) = " << y << std::endl;

    y = std::cos(M_PI_2 - std::asin(x));
    std::cout << "cos(pi/2 - asin(" << x << ")) = " << y << std::endl;

    return 0;
}
```

Output:

```
Enter a real value 'x': 0.32
sin(0.32) + 1 = 1.31457
sin(asin(0.32)) = 0.32
cos(pi/2 - asin(0.32)) = 0.32
```


Hyperbolic, exponential and logarithmic functions

- Other functions in the `cmath` library are:

```
double cosh( double x );
double sinh( double x );
double tanh( double x );
double acosh( double x );
double asinh( double x );
double atanh( double x );

double exp( double x );
double log( double x );     // calculates ln(x)
double log10( double x );   // calculates log  (x)
                            //               10
```


Other functions

- Other functions in the `cmath` library are:

```
                                         //            y
double pow( double x, double y );        // Computes x
double sqrt( double x );     // The square root of x
double cbrt( double x );     // The cube root of x
```

Calling `sqrt(3.2)` is the same as `pow(3.2, 0.5)`
Calling `cbrt(3.2)` is the same as `pow(3.2, 1.0/3.0)`

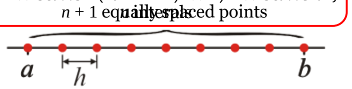
```
                                  //      _______
                                  //     / 2    2
double hypot( double x, double y ) //  \/  x  + y

double ceil( double x );  // Least integer greater than or equal to x
double floor( double x ); // Greatest integer less than or equal to x
double trunc( double x ); // Remove the fractional part of x
double round( double x ); // Round x to the nearest integer

double abs( double x );
```

A nice example

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

    double b{};
    std::cout << "Enter a real value 'b': ";
    std::cin >> b;

    int n{};
    std::cout << "How many intervals? ";
    std::cin >> n;

    double h{(b - a)/n};

    for ( int k{0}; k <= n; ++k ) {
        std::cout << "(" <<          a + k*h   << ", "
                         << std::sin( a + k*h ) << ")" << std::endl;
    }

    return 0;
}
```

Output:

```
Enter a real value 'a': 2.5
Enter a real value 'b': 3.7
How many intervals 'n': 6

(2.5, 0.598472)
(2.7, 0.42738)
(2.9, 0.239249)
(3.1, 0.0415807)
(3.3, -0.157746)
(3.5, -0.350783)
(3.7, -0.529836)
```



An important takeaway

- To use a function, all you need to know is:
  - The function declaration, which tells you
    - How many and the types of the arguments
    - The type of the return value
  - A description of what the function is supposed to accomplish
- If the actual behavior of the function does not match the description,
  this is a *bug*; that is, a situation where the expected returned value does not match the actual returned value
- As long as you trust the author of the function,
  you don't have to know how the function is implemented
  - You probably shouldn't care, either…

Summary

- Following this lesson, you now:
  - Understand that libraries in C++ can contain global variables and functions
  - Know how to look at the function declaration and call that function
  - Know what you can do with the value returned by a function
  - Are aware of the C math library and some of its contents
  - Know that you can use a function
    without understanding how it was implemented

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

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.