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

ECE 150 *Fundamentals of Programming*

Assertions

ECE150

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

Outline

- This is the first in a sequence of six topics on
  - C assertions
  - Code development strategies
  - Testing
  - Commenting your code
  - Using print statements for debugging
  - Using tracing for debugging

Outline

- In this tutorial, we will:
  - Describe the `assert` "function"
  - Consider its uses
  - See how to turn assertions off

C-style assertions

- Up to this point, we have only executed functions and dealt with all possible arguments
  - For example, the factorial is not defined for negative integers
    - We, however, returned zero
  - Also, we have arbitrarily executed the alternative body in a conditional statement
    - Can we check to make sure that the conditions are as expected?

C-style assertions

- An assertion is a "function" that takes a Boolean-valued condition
  - If the condition is `true`, the program continues executing
  - If the condition is `false`, the program terminates with an error

- For example:

```
int factorial( int n ) {
    assert( n >= 0 );

    int result{1};

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

    return result;
}
```

It is actually a *macro*, which is beyond the scope of this course

C-style assertions

- To use the assert function, you must include the C assert library:

```
#include <cassert>
```

- Suppose we have the following program:

```
#include <iostream>
#include <cassert>

// Function declarations
int main();
int factorial( int n );

// Function definitions
int main() {
    std::cout << factorial( 10 ) << std::endl;
    std::cout << factorial(  0 ) << std::endl;
    std::cout << factorial( -2 ) << std::endl;

    return 0;
}
```

Output:

```
3628800
1
a.out: example.cpp:18:
  int factorial(int):
    Assertion `n >= 0' failed.
```

C-style assertions

- Consider this program:

```
#include <iostream>
#include <cassert>

// Function declarations
int main();
int factorial( int n );

// Function definitions
int main() {
    for ( int k{0}; k <= 17; ++k ) {
        std::cout << k << "! = "
                  << factorial( k ) << std::endl;
    }

    return 0;
}
```

Output:

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 1932053504
14! = 1278945280
15! = 2004310016
16! = 2004189184
17! = -288522240
```

C-style assertions

- Thus, a better implementation of the factorial function is:

```
int factorial( int n ) {
    assert( (n >= 0) && (n <= 12) );

    int result{1};

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

    return result;
}
```

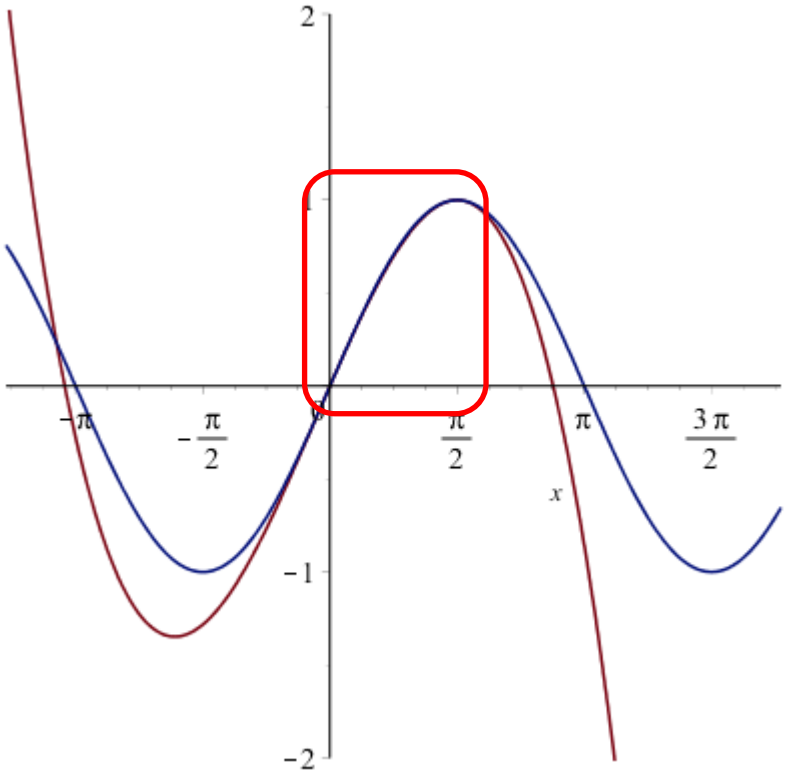
Example

- Previously, we introduced a *spline*

$$4\frac{x^2}{\pi^2}\left(x-\frac{4}{\pi}x-\pi+3\right)+x$$

- When plotted next to the sine function, it's a good approximation if

$$0 \le x \le \frac{\pi}{2}$$



Example

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

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

int main() {
    std::cout <<   my_sin( 0.5 ) << std::endl;
    std::cout << std::sin( 0.5 ) << std::endl;
    std::cout <<   my_sin( 1.6 ) << std::endl;
    return 0;
}

double my_sin( double x ) {
    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;
}
```

Output:

```
0.471811
0.479426
a.out: example.cpp:18:
  double my_sin(double):
    Assertion `(x >= 0.0) && (x <= M_PI_2)' failed.
```

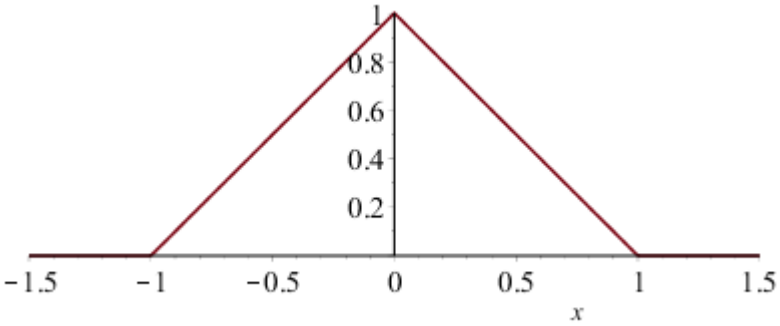
Checking conditional statements

- Suppose you have a cascading conditional statement
  - It may be useful to ensure that the condition in the complementary alternative body is what is expected

```
#include <cassert>

// Function declarations
double tent( double x );

// Function definitions
double tent( double x ) {
    if ( (x <= -1) || (x >= 1) ) {
        return 0.0;
    } else if ( x <= 0 ) {
        return x + 1.0;
    } else {
        assert( (x > 0.0) && (x < 1.0) );
        return 1.0 - x;
    }
}
```



Summary

- Following this lesson, you now:
  - Know how to use the `assert` "function"
  - Understand it can be used to:
    - The arguments passed to a function are as expected
    - Values are as expected when executing code
  - Understand that assertions are never needed in this course
    - They only help you catch errors in your own code

References

[1] Wikipedia: https://en.wikipedia.org/wiki/Assert.h

[2] Cplusplus.com
http://www.cplusplus.com/reference/cassert/

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.