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

ECE 150 *Fundamentals of Programming*

Writing tests

ECE150

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

Outline

- This is the third 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 topic, we will:
  - Describe how to write a test
  - Explain why tests should be written first
  - Look at some examples where we write tests for specific functions
  - Emphasize that all code should be executed by at least one test

Testing your code

- When you are given a project or assignment,
  you will be given a description and requirements
  - Initially, we will give you many if not all appropriate tests
  - You should, however, always consider
    "What is required, and how can I test this?"

Median of three

- Suppose you are asked to author a median-of-three function
  - The median is the middle number

- The function declaration is:

```
double median( double x, double y, double z );
```

Median of three

- In your `main()` function, you could now include a test:

```
#include <iostream>

// Function declarations
int main();
double median( double x, double y, double z );

// Function definitions
int main() {
    std::cout << median( 1.2, 3.5, 7.9 ) << " = 3.5" << std::endl;
    return 0;
}

double median( double x, double y, double z ) {
    // Your implementation of this function
    return 0.0;
}
```

Output:

```
3.5 = 3.5
```

Median of three

- One test, however, is not enough
  - First, the median could be in any location:

```
std::cout << median(  5.4,  3.5,  7.9 ) << " =  5.4" << std::endl;
std::cout << median( -1.2,  3.5,  7.9 ) << " =  3.5" << std::endl;
std::cout << median(  8.2, -8.5, -4.5 ) << " = -4.5" << std::endl;
```

    - Note that both positive and negative numbers are used
      - Don't just favor positive values because they're easier to type

  - Next, switch the order of the other two entries:

```
std::cout << median(  6.4,  7.5,  3.9 ) << " =  6.4" << std::endl;
std::cout << median( 11.2,  1.5, -8.5 ) << " =  1.5" << std::endl;
std::cout << median( -8.3, 22.5, -2.5 ) << " = -2.5" << std::endl;
```

Median of three

- Next, will your code execute correctly if two of the or all three arguments are equal?

```
std::cout << median(  1.1,  1.1,  1.1 ) << " =  1.1" << std::endl;

std::cout << median( -5.4, -9.5, -5.4 ) << " = -5.4" << std::endl;
std::cout << median( -9.2,  7.5,  7.5 ) << " =  7.5" << std::endl;
std::cout << median(  8.2,  8.2, -4.5 ) << " =  8.2" << std::endl;

std::cout << median( -1.5,  8.9, -1.5 ) << " = -1.5" << std::endl;
std::cout << median( 19.2, -1.5, -1.5 ) << " = -1.5" << std::endl;
std::cout << median(  8.6,  8.6, 99.5 ) << " =  8.6" << std::endl;
```

  - Notice that the values are being changed with each example?

Median of three

- Finally, it's not a bad idea to test some really extreme cases:

```
std::cout << median( -5.092e73,  3.5113e99, -9.283e-82 )
          << " = -5.092e73" << std::endl;
```

Median of three

- Thus, here is our new `main()` function:

```
int main() {
    std::cout << median(  5.4,  3.5,  7.9 ) << " =  5.4" << std::endl;
    std::cout << median( -1.2,  3.5,  7.9 ) << " =  3.5" << std::endl;
    std::cout << median(  8.2, -8.5, -4.5 ) << " = -4.5" << std::endl;

    std::cout << median(  6.4,  7.5,  3.9 ) << " =  6.4" << std::endl;
    std::cout << median( 11.2,  1.5, -8.5 ) << " =  1.5" << std::endl;
    std::cout << median( -8.3, 22.5, -2.5 ) << " = -2.5" << std::endl;

    std::cout << median(  1.1,  1.1,  1.1 ) << " =  1.1" << std::endl;

    std::cout << median( -5.4, -9.5, -5.4 ) << " = -5.4" << std::endl;
    std::cout << median( -9.2,  7.5,  7.5 ) << " =  7.5" << std::endl;
    std::cout << median(  8.2,  8.2, -4.5 ) << " =  8.2" << std::endl;

    std::cout << median( -1.5,  8.9, -1.5 ) << " = -1.5" << std::endl;
    std::cout << median( 19.2, -1.5, -1.5 ) << " = -1.5" << std::endl;
    std::cout << median(  8.6,  8.6, 99.5 ) << " =  8.6" << std::endl;

    std::cout << median( -5.092e73,  3.5113e99, -9.283e-82 )
              << " = -5.092e73" << std::endl;
    return 0;
}
```

Median of three

- If you execute your code, it should compile:

Output:

```
0 =  5.4
0 =  3.5
0 = -4.5
0 =  5.4
0 =  3.5
0 = -4.5
0 =  1.1
0 = -5.4
0 =  7.5
0 =  8.2
0 = -5.4
0 = -7.5
0 =  8.2
0 = -5.092e73
```

Median of three

- As you implement your function,
  more and more of the outputs should appear as expected

Output:

```
5.4 =  5.4
3.5 =  3.5
-4.5 = -4.5
5.4 =  5.4
3.5 =  3.5
-4.5 = -4.5
1.1 =  1.1
-5.4 = -5.4
7.5 =  7.5
8.2 =  8.2
-5.4 = -5.4
-7.5 = -7.5
8.2 =  8.2
-9.283e-82 = -5.092e73
```

Median of three

- If the output differs from what is expected, there are always two possibilities:
  - There is a bug in the source code
  - There is a bug in the test

- What is wrong here?

```
std::cout << median( -5.092e73,  3.5113e99, -9.283e-82 )
          << " = -5.092e73" << std::endl;
```

- The correct test is:

```
std::cout << median( -5.092e73,  3.5113e99, -9.283e-82 )
          << " = -9.283e-82" << std::endl;
```

Testing your code

- Why write the tests first?
  - If you write your source code first,
    your source code will influence the tests you write
  - If you made a mistake in your reasoning while authoring your code,
    you may make the exact same mistake when authoring the tests

- The tests should be written based on the specifications and requirements

Testing your code

- Here is a strategy:

  Working with at least one other student, take turns for each project or assignment, where one student writes the tests before starting to author a solution.

  Share the test cases with your peers.

- In ECE 250, on occasion, one student would author a 100-line and even a 1000-line test for all students in the course
  - In one case, the test missed one interesting *edge* case,

    so almost all students got that edge case wrong
  - In one case, the class asked if that student could get a bonus

Testing your code

- In this course, you will never be penalized for sharing tests
  - Sharing tests is encouraged

Never share your solutions to the assignments or projects!!!

Edge cases

- An *edge* case is a situation when one parameter takes on a specific value that approaches a boundary
  - Recall the example of reversing the digits of a number:
    - Given 9512, output `9512|2159`
    - Given -42, output `-42|24-`
  - My source code would not work given 0, it would print `0|`
    - Had I not thought to test this edge case,
      my tests would have all passed and I would be oblivious
    - Recall that in the end, I dealt with this case separately:

```
void reverse( int n ) {
    if ( n == 0 ) {
        std::cout << "0|0" <<std::endl;
        return;
    }
```

Greatest common divisor

- What tests would be author for the greatest common divisor?
  - gcd(n, n) = n
  - gcd(n, $-n$) = n
  - gcd($-n$, $-n$) = n
  - gcd($-n$, n) = n
  - gcd(n, $5n$) = n
  - gcd(n, $-12n$) = n
  - gcd(n, 0) = n
  - gcd(0, 0) = 0
  - gcd(m, n) = gcd(n, m) for various pairs of m and n
  - Find cases when gcd(m, n) = 1

- Check when m and n are prime and composite
  - Find some very large prime numbers and use them in your tests

Ensure all code is tested

- When you finish writing your code,
  make sure that each line of code is executed in at least one test
  - This is a way of checking your test cases
- For example, this code finds them minimum of three values:

```
int min( int a, int b, int c ) {
    if ( a <= b ) {
        if ( a <= c ) {
            return a;
        } else {
            return c;
        }
    } else {
        if ( b <= c ) {
            return b;
        } else {
            return c;
        }
    }
}
```



Summary

- Following this lesson, you now:
  - Know you should author tests
    - Tests should be written before the code is written
    - Work with your peers
  - Ensure that all code is tested at least once

References

[1] Wikipedia:
https://en.wikipedia.org/wiki/Therac-25
https://en.wikipedia.org/wiki/Tacoma_Narrows_Bridge_(1940)
https://en.wikipedia.org/wiki/Citigroup_Center#Engineering_crisis_of_1978
https://en.wikipedia.org/wiki/Software_testing

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.