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

ECE 150 *Fundamentals of Programming*

Code-development strategies

Douglas Wilhelm Harder, M.Math.
Prof. Hiren Patel, Ph.D.
hiren.patel@uwaterloo.ca dwharder@uwaterloo.ca

Outline

- In this lesson, we will:
  - Discuss how most novice programmers will tackle assignments
  - Provide a logical and methodical approach
  - A code *skeleton* allows you to focus on one function at a time
    - Compile-time errors will be localized to the code you are authoring

Code development

- In the course assignments, you will be asked to implement a number of functions, and later, structures and classes
- We have now looked at functions, conditional statements and repetition statements
  - We will now look at strategies for developing a code base

Code development

- Suppose you are asked in an assignment to implement the following five functions that
  1. implements an absolute-value function,
  2. approximates the tangent function on the open interval $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$,
  3. implements the greatest-common divisor algorithm,
  4. implements the least-common multiple algorithm, and
  5. asks the user for a name, e.g., "Bob" and then prints "Hi Bob!".

together with a `main()` function that uses these functions

Code development

- Many first-year students and novice programmers will do the following:
  - They will write all the code, try compiling and be left with a bewildering number of error messages
  - Problem: errors such as forgetting semi-colons or closing parentheses or braces can often give messages that are far away from the actual issue
  - Problem: for a novice programmer, seeing page after page of error messages can be intimidating and unnecessarily frustrating

Code development

- Best approach is to write a *skeleton*, and make sure it compiles:
  - List all of the function declarations
    - If you are not given names for the functions, come up with names that are appropriate and descriptive
  - Create *skeleton* implementations of each of the functions that simply return a default value
  - Make sure the skeleton compiles
  - Then start working on one function at a time, and
    - Make sure the code still compiles, and
    - Try to test the implementation in `main()` to ensure it works correctly

Code development

- You will now create your skeleton:

```
#include <iostream>

// Function declarations
int main();
double my_abs( double x );
double my_tan( double x );
int gcd( int m, int n );
int lcm( int m, int n );
void print_hi();

// Function definitions
int main() {
    return 0;
}
```

You don't have to, but it's a good idea to keep the function definitions in the same order as the function declarations

Code development

- For each function, just have the definition return the default value

```
double my_abs( double x ) {
    return 0.0;
}

double my_tan( double x ) {
    return 0.0;
}

int gcd( int m, int n ) {
    return 0;
}

int lcm( int m, int n ) {
    return 0;
}

// Nothing to return here--at the end of the function body, it returns
void print_hi() {
}
```

Code development

- This code should compile, even if it does nothing
  - If there are compile-time errors, fix them first
- Next, implement the `main()` function just to test everything

```
// Function definitions
int main() {
    std::cout << "gcd(15708, 23205) = 357: " << gcd( 15708, 23205 )
              << std::endl;
    std::cout << "lcm(15708, 23205) = 1021020: " << lcm( 15708, 23205 )
              << std::endl;
    std::cout << "|-3.2| = 3.2: " << my_abs(-3.2) << std::endl;
    std::cout << "tan(0.01) = 0.0100003333: " << my_tan(0.01)
              << std::endl;

    print_hi();

    return 0;
}
```

Code development

- If you tried compiling and executing this, it would be nonsensical:

```
gcd(15708, 23205) = 357: 0
lcm(15708, 23205) = 1021020: 0
|-3.2| = 3.2: 0
tan(0.01) = 0.0100003333: 0
```

- However, it does compile!

Code development

- Next, implement one function at a time, and make sure it works
  - Don't start `lcm` until you get `gcd` working
  - If you think that the absolute-value function is easier, start with it, but make sure it works before you go onto the next
- Many students have an attitude combining thoughts like:

  "This is too easy…"
  "I won't make any mistakes!"
  "I want to get this #$@&%*! done and over with!!!"
  "Compiling wastes my time…"

- And one in one hundred students will be right on all counts
  - The rest, however, may ignore these suggestions and in the end, waste more time than they would have had to… ☹

Summary

- Following this lesson, you now:
  - Understand that writing everything at once is likely going to result in longer development time
  - Know that starting by writing a skeleton is easy enough
  - Understand that by writing one function at a time, you can
    - First, focus on any compile-time errors your implementation has
    - Next, ensure your implementation works correctly
- The course web site presents an example that prints the volume and surface areas of the five Platonic solids:

  https://ece.uwaterloo.ca/~ece150/Lecture_materials/1.10/

References

[1] Wikipedia
https://en.wikipedia.org/wiki/Skeleton_(computer_programming)

Acknowledgments

Proof read by Dr. Thomas McConkey and Charlie Liu.

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