05. Syntax Patterns, Code Style, Format Debugging

CPSC 120: Introduction to Programming
Kevin A. Wortman ~ CSU Fullerton

Agenda

0. Sign-in sheet
1. Q&A
    a. Monday Labs will do Linux install and abbreviated Lab 2 today
2. Syntax Patterns
3. Code Style
4. Format Debugging

1. Q&A

Q&A

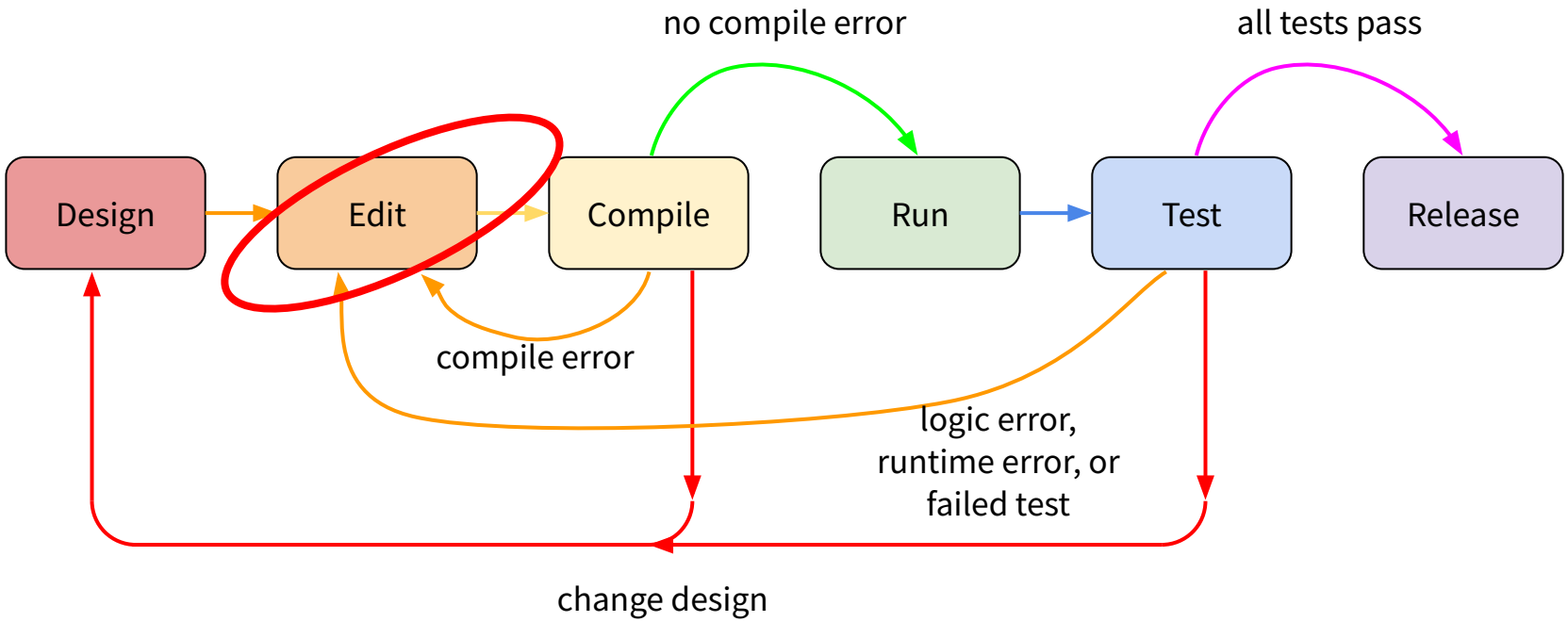
Let’s hear your questions about...

- This week’s Lab
- Linux
- Any other issues

Reminder: write these questions in your notebook during lab

2. Syntax Patterns

The Development Cycle



Syntax and Semantics

	Syntax	Semantics
In general	source code **structure**	source code **meaning**
`std::cout << “Hello World!”;`	cout, then <<, then “Hello world!”, then ;	print out Hello World!

Syntax Patterns

Same as cppreference.com:

Syntax is...	Notation
verbatim (write exactly as-is)	**bold**
fill-in-the-blank	*italics*
optional	has(optional)
may be repeated	ellipsis...

Pattern of a Source File

source-file:

directive, declaration, or definition...

Example: chrome_history_client.cc

Semantics:

- The compiler processes each *directive, declaration,* or *definition* in top-to-bottom order.

Hello World

```
// our hello world program
#include <iostream>
```

directive

```
int main(int argc, char* argv[]) {
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

definition

Pattern for a Program

program:

a program is one or more *source files* that contains exactly one *main function definition*

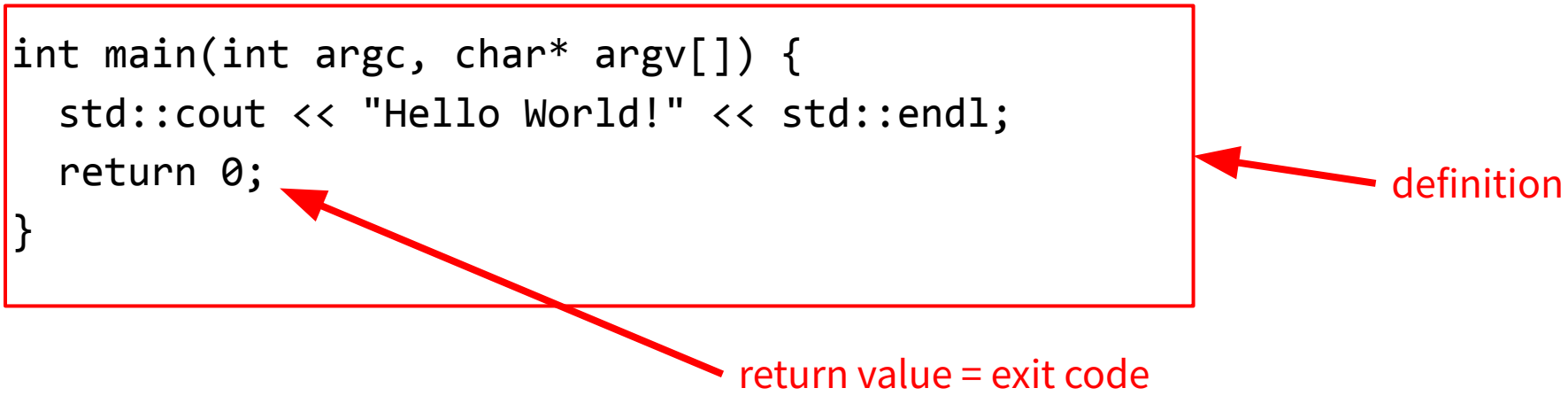
Semantics:

- The program starts by executing **main**
- The return value of **main** is the exit code of the program

Hello World

```
// our hello world program
#include <iostream>

int main(int argc, char* argv[]) {
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```



Syntax Categories

Category	Semantics	Example
directive	orders the compiler to compile in a certain way	`#include <iostream>`
declaration	introduce the name of a variable, function, or data type	`int increase(int value);`

definition	declaration that also includes the body of a function or data type	`int decrease(int value) {`
		`  return value - 1;`
		`}`

statement	perform one step of an algorithm inside a function body	`std::cout << “Hello world”;`
expression	inside a statement, use operators to calculate a value	`(price + tax)`

Pattern for Main Function Definition

definition:

```
int main(int argc, char* argv[]) {
  statement...
}
```

Semantics:

- Execute *statement...* in **top-to-bottom order**

Fill-in-the-Blanks are Interchangeable

- You can fill a blank with **any** syntax of the matching type
- In

  directive, declaration, or definition...

  you can fill in any kind of *directive* or *declaration* or *definition*

- In

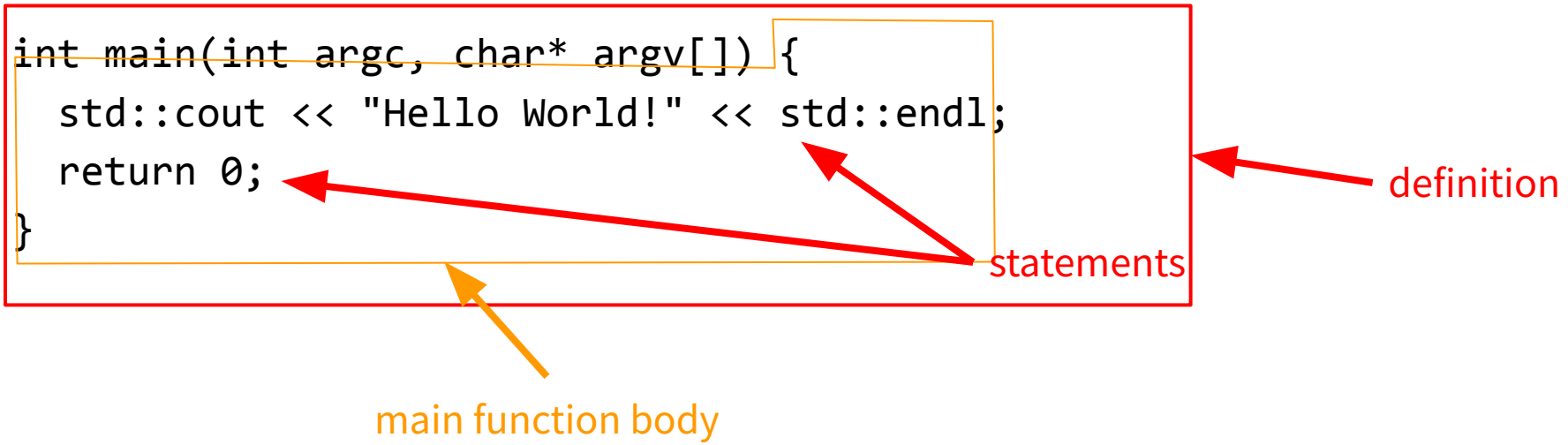
  statement...

  you can fill in any kind of *statement*

Hello World

```
// our hello world program
#include <iostream>

int main(int argc, char* argv[]) {
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```



Whitespace

- **Whitespace**: invisible formatting (space, tab, newline)
- Ignored by compiler
- Can go in between other syntax

Comments

- **Comment**: text in source code that is ignored by the compiler
- Purpose: notes, rationale, authorship, copyright
- Audience: other programmers, your future self
- Like whitespace, is allowed anywhere

comment:

// *text...*

Example: chrome_history_client.cc

Semantics:

- Compiler ignores // and *text...*

3. Code Style

Clean Code

“Clean code is code that is easy to understand and easy to change.” --Carl Vuorinen

- Source code is for **human** consumption
- Code lifetime
  - **write once**
  - **read many times**
  - Example: chrome_history_client.cc
- Clarity matters
  - Hard for you to debug unclear code
  - Coworkers
- Valued in job market

Clean versus Unclean Whitespace

```
int main(int argc, char* argv[]) {
  std::cout << "Hello World!" << std::endl;
  return 0;
}

int main(int argc,char* argv[]){std::cout<<"Hello
World!"<<std::endl;return 0;}
```

Style Guide

- **Style guide**: defines clean/unclean code
- Living document
- We use Google C++ Style Guide
- Common issues in lab 2:
  - Horizontal Whitespace
  - Vertical Whitespace
  - Function Declarations and Definitions (curly brace { placement)

4. Format Debugging

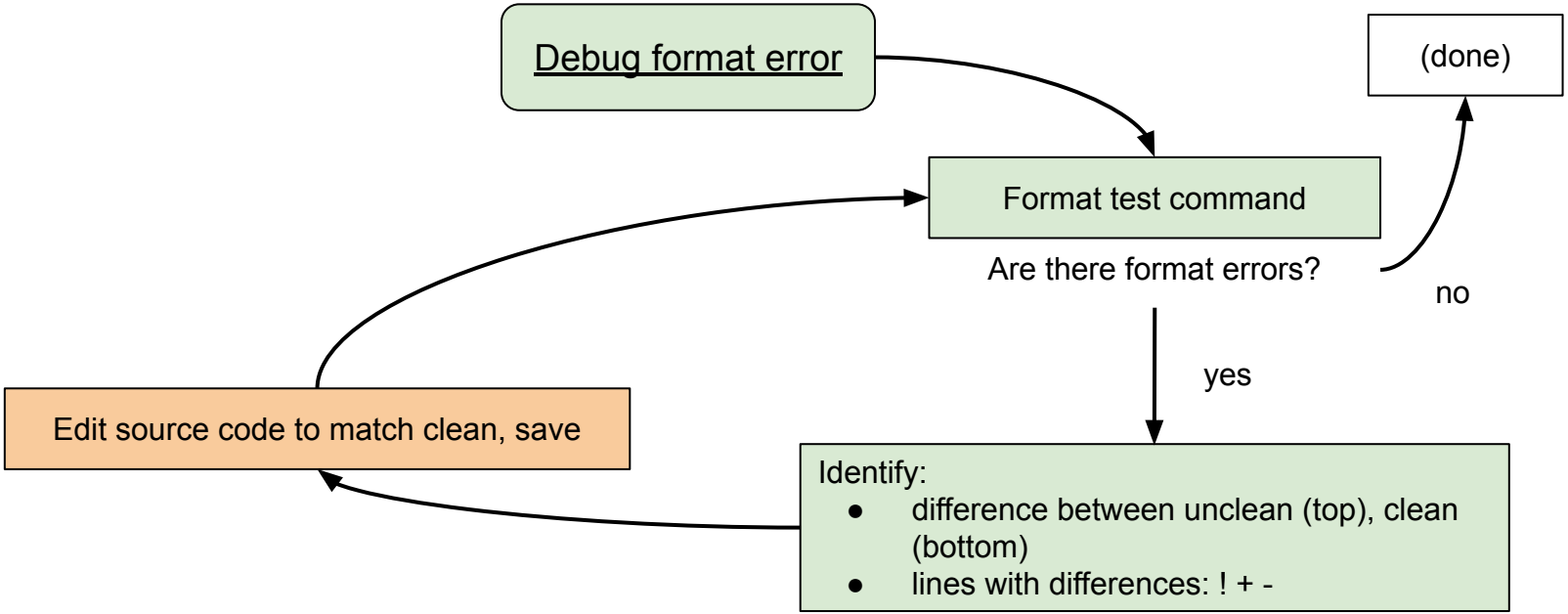
Ideal Division of Labor

- **Business Logic:** the human meaning of algorithm data
- Programs
  - **Cannot** understand business logic or design algorithms
  - Can perform tedious, repetitive work flawlessly, quickly, cheaply
- Humans
  - **Can** understand business logic and design algorithms
  - Busy-work is tedious, error-prone, expensive
- Division of Labor Best Practice
  - Humans think about business logic and algorithms
  - Computer programs do repetitive work

Automating Clean Code

- **Focus of lab 2**
- Program (not person) checks code
- Corresponds to Google C++ Style Guide
- clang-format: checks syntax
  - whitespace, variable names, ...
- **linter** (clang-tidy): checks logic errors
  - coming soon

Debugging Format Errors



No Format Errors

```
$ ./check_formatting
2023-02-03 17:24:13,465 - INFO - Checking format for file:
/home/csuftitan/cpsc-120-solution-lab-02/part-1/fahrenheit_to_celsius.cc
2023-02-03 17:24:15,422 - INFO - 😀 Formatting looks pretty good! 🥳
2023-02-03 17:24:15,422 - INFO - This is not an auto-grader.
2023-02-03 17:24:15,422 - INFO - Make sure you followed all the instructions and
requirements.
```

Format Errors

```
  int main(int argc, char const *argv[]) {
! std::cout << "Hello World!";
      return 0;
  }

--- 16,22 ----

  using namespace std;

  int main(int argc, char const *argv[]) {
!   std::cout << "Hello World!";
      return 0;
  }

2023-02-03 17:36:54,726 - ERROR - 🤯😳😤😫😡
2023-02-03 17:36:54,726 - ERROR - Your formatting doesn't conform to the Google C++ style.
2023-02-03 17:36:54,726 - ERROR - Use the output from this program to help guide you.
2023-02-03 17:36:54,726 - ERROR - If you get stuck, ask your instructor for help.
2023-02-03 17:36:54,726 - ERROR - Remember, you can find the Google C++ style online at
https://google.github.io/styleguide/cppguide.html.
```

Contextual Diff

- GNU Diffutils: programs for identifying differences between files
- Contextual Diff: prints differences **with surrounding context**
- Compares **unclean source** to hypothetical **cleaned source**
- **Hunk** of differences: area that differs

Contextual Diff Format

```
***************
*** first-unclean-line, last-unclean-line ****
  unclean-line...
--- first-clean-line, last-clean-line ----
  clean-line...
```

Left column:

!	lines differ
+	line added to unclean
-	line deleted from unclean

Example: Contextual Diff Output

```
2023-02-03 17:36:54,718 - ERROR - Error: Formatting needs improvement.
2023-02-03 17:36:54,726 - WARNING - Contextual Diff
*** Student Submission (Yours)

--- Correct Format

***************

*** 16,22 ****

  using namespace std;

  int main(int argc, char const *argv[]) {
! std::cout << "Hello World!";
       return 0;
  }

--- 16,22 ----

  using namespace std;

  int main(int argc, char const *argv[]) {
!   std::cout << "Hello World!";
       return 0;
```

Recap: Debugging Format Errors