05. Format Debugging, Variables, Literals, Input/Output

CPSC 120: Introduction to Programming
Pratishtha Soni~ CSU Fullerton

Agenda

0. Sign-in sheet
1. Technical Q&A
2. Format Debugging
3. Variables and Literal Expressions
4. Input/Output

1. Technical Q&A

Technical Q&A

Let’s hear your noted questions about...

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

Reminder: write these questions in your notebook during lab

2. 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

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

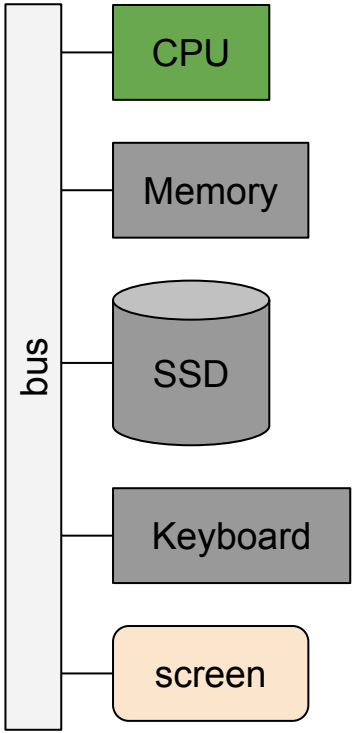
```
2023-02-03 17:36:54,718 - ERROR - Error: Formatting needs improvement.
2023-02-03 17:36:54,726 - WARNING - Contextual Diff
*** Student Submission (Yours)

--- 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;
```

Debugging Format Errors

1. Run format check
2. Identify lines with differences; left column is one of: ! + -
3. Identify difference between unclean(top) and clean (bottom) source
4. Edit source code to match clean
5. Save, go back to step 1

3. Variables and Literal Expressions

Review: Computer Architecture

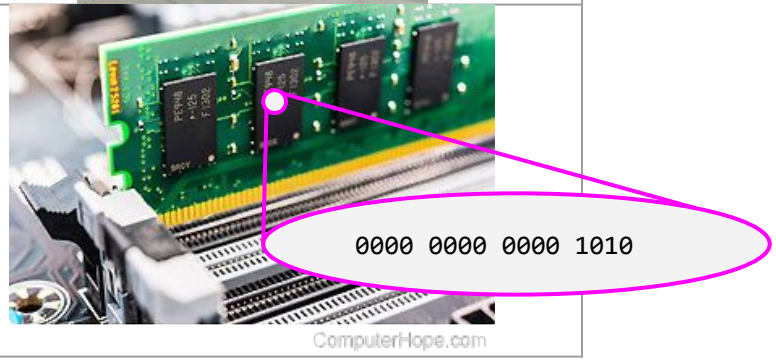


A **program:**

- Is made up of CPU instructions
- Tells the CPU to perform calculations and move data between memory, SSD, keyboard, screen, etc.
- Corresponds to an algorithm
- INPUT from keyboard or SSD
- OUTPUT to screen or SSD

This is all that programs do!

Objects and Variables

Kind of Object	Name	Picture
building	Engineering Building (E)	
piece of data stored in memory	variable `int score{ 10 };`	

Objects and Variable Vocabulary

- **Object** (n): region of memory that stores a piece of information
- **Variable** (n): a name for an object in source code
- **Declare** (v): create a variable
- **Initialize** (v): store a particular object in a variable

Syntax: Variable Declaration and Initialization

statement:

data-type identifier **{** *expression* **};**

Examples:

```
int count{ 0 };

double temperature{ 98.6 };
```

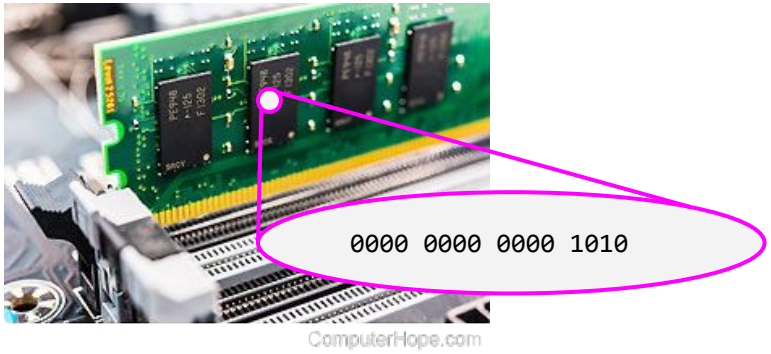
Semantics:

- Declare variable with name *identifier* and type *data-type*
- Initialize *identifier* to store the result of evaluating *expression*

Next: how to fill in *data-type*, *expression*, *identifier*

Data Types

- **Data type**
  - “Type” for short
  - Format for storing an object in memory
  - Defined operations in source code
- Will explore many data types
- For today, just two...
- **`int`**: integer (whole number)
- **`double`**: double-precision floating-point number (decimal number)




```
int count{ 0 };

double temperature{ 98.6 };
```

Expressions

- **Expression:** combination of variables, literals, operators, and function calls that may be **evaluated** to produce a **result**
- Result has a specific **type** and **value**
- **Literal** expression: value is written explicitly in source code

Example Expression	Result Type	Result Value
`107.3`	`double`	`107.3`
`100 - 1`	`int`	`99`
`temperature`	`double` (same as temperature)	`98.6`
`temperature + 2.0`	`double` (same as temperature)	`100.6`

Syntax: Integer Literal Expression

literal integers

expression:

-(optional) *digit...*

```
1
2   #include <iostream>
3
4   int main(int argc, char* argv[]) {
5       int this_year{ 2022 },
6           birth_year{ 1956 },
7           age{ this_year - birth_year };
8       std::cout << "Age is " << age << "\n";
9       return 0;
10  }
```

Semantics:

- digits (and optional - sign) are result in a value of type `int`

Syntax: Double Literal Expression

literal double

expression:

-(optional) *whole-digit...* . *decimal-digit...*

Example:

```
double temperature{ 98.6 };

double slope { -2.1 };
```

Semantics:

- whole part, decimal part, and optional - sign result in a value of type `double`

Syntax: Identifier Expression (id-expression)

expression:

identifier

Semantics:

- *identifier* must be a declared variable, otherwise compile error
- result
  - type is same as variable *identifier*
  - value is current contents of variable *identifier*

```
1
2   #include <iostream>
3
4   int main(int argc, char* argv[]) {
5       int this_year{ 2022 },
6           birth_year{ 1956 },
7           age{ this_year - birth_year };
8       std::cout << "Age is " << age << "\n";
9       return 0;
10  }
```

Syntax: Multiple Declaration and Initialization

statement:

data-type identifier-and-obj...;

identifier-and-obj:

identifier { *expression* }

Examples:

```
int p1_score{5}, p2_score{6};
```

Semantics:

- Each *identifier* is declared, and initialized with the result of *expression*
- All variables have the same *data-type*

Uninitialized Variables

- Initial objects are technically optional
- Style guide: **always** initialize variables
- **Uninitialized variable**: variable that has not been initialized
  - Contents is **undefined**
  - Junk / “random”
- **Undefined behavior**: no rule for what compiler, CPU must do
  - Always a bug
  - May appear a runtime error or logic error
  - May not appear
- **Programs should not have**
  - **undefined behavior**
  - **uninitialized variables**

Example: Undefined Behavior

```
2   #include <iostream>
3
4   int main(int argc, char* argv[]) {
5       int year;
6       std::cout << "Year is " << year << "\n";
7       return 0;
8   }
```

Output could be any of:

Year is 0

Year is -80401

Year is 2147483647

Syntax: Variable Declaration (w/o Init.)

statement:

data-type identifier... **;**

Examples:

```
double grade;

int hours, minutes;
```

Semantics:

- Declare variable(s) with name *identifier* and type *data type*
- Variables are **uninitialized**
- Style guide violation
- **Do not do this**
- Always initialize variables

4. Input/Output

Standard Input/Output

- **Standard output**: text printed by program
- **Standard input**: text typed into program
- **cout**: standard output object
  - “c” for character
- **cin**: standard input object

standard output

```
$ git clone https://github.com/cpsc-pilot-fall-2022/cpsc-120-prompt-lab-02.git
Cloning into 'cpsc-120-prompt-lab-02'...
remote: Enumerating objects: 167, done.
remote: Counting objects: 100% (167/167), done.
remote: Compressing objects: 100% (136/136), done.
remote: Total 167 (delta 23), reused 164 (delta 20), pack-reused 0
Receiving objects: 100% (167/167), 654.86 KiB | 1.31 MiB/s, done.
Resolving deltas: 100% (23/23), done.
```

Syntax: cout Expression

expression:

std::cout *insert-expression...*

insert-expression:

<< *expr*

- In left-to-right order, each *expr:*
  - Is evaluated to produce a result
  - Result type must be printable; otherwise compile error
  - Result value is printed to standard output
- `int` and `double` are printable

Examples:

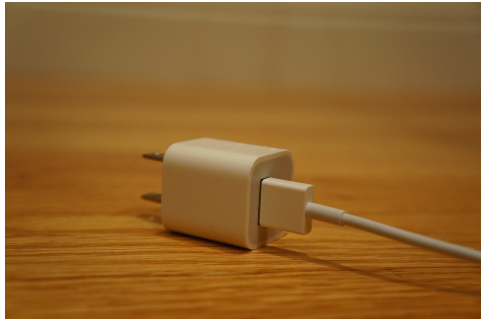
```
std::cout << 7
std::cout << “Hello” << “ there”
std::cout << (2 * 10)
```

Review: Pattern for Main Function Definition

definition:

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

- need to fill in the blank with a *statement*
- But cout expression is an *expression*
- **Expression statement**: statement that holds an expression
  - Adapter
  - Allows an expression to “count” as a statement



Syntax: Expression Statement

statement:

expr ;

Semantics:

- Evaluate *expr*
- Discard the result
- (That's all)

Examples:

```
std::cout << "Hi" << " there";
```

```
int score{0};
score + 1;  // has no effect
            // probably a bug
```

Example: cout

```
1
2   #include <iostream>
3
4   int main(int argc, char* argv[]) {
5       int year{ 2022 };
6       std::cout << "Year is " << year << "\n";
7       return 0;
8   }
```

Syntax: cin Expression

expression:

std::cin *extract-expression...*

extract-expression:

>> *variable*

In left-to-right order, for each *variable:*

- If cin already **failed**: do nothing
- Otherwise:
  - Skip whitespace, read characters from standard input
  - If they represent an object of *variable*'s type: store that object in *variable*
  - Otherwise: cin is **failed**; leave *variable* unchanged

cin expression in expression statements:

```
int year{ 0 };

std::cout << "Enter year: ";

std::cin >> year;
```

Example: cin

```
2   #include <iostream>
3
4   int main(int argc, char* argv[]) {
5       int birth_year{ 0 }, this_year{ 0 };
6       std::cout << "Enter birth year: ";
7       std::cin >> birth_year;
8       std::cout << "Enter this year: ";
9       std::cin >> this_year;
10      std::cout << "In " << this_year << ", a person"
11                << " born in " << birth_year
12                << " is " << (this_year - birth_year)
13                << " years old.\n";
14      return 0;
15  }
```

Valid input:

```
$ ./a.out
Enter birth year: 1961
Enter this year: 2022
In 2022, a person born in 1961 is 61 years old.
```

Failed input:

```
$ ./a.out
Enter birth year: snake
Enter this year: In 0, a person born in 0 is 0
years old.
```

cin/cout Pitfalls

- Keep operators straight: std::cout << , std::cin >>
- cin only works with variables
  - std::cout << “Enter a number:”; OK
  - std::cin >> “Enter a number:”; compile error
- << or >> between each part
  - std::cout << “Hello” “there”; compile error
- Semicolon at end
  - std::cout << “Hello” << “ there” compile error