06. Variables, Literals, Input/Output

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

Agenda

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

1. Q&A

Q&A

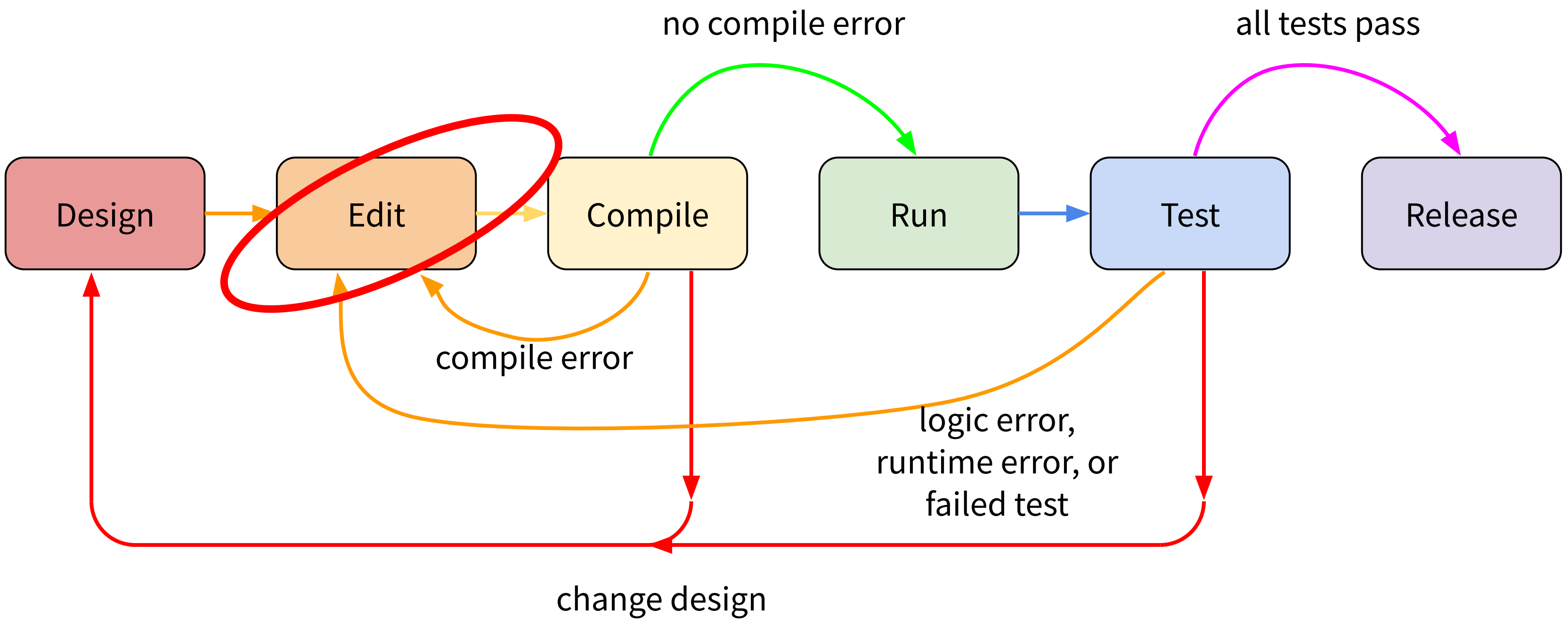
Let’s hear your questions about...

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

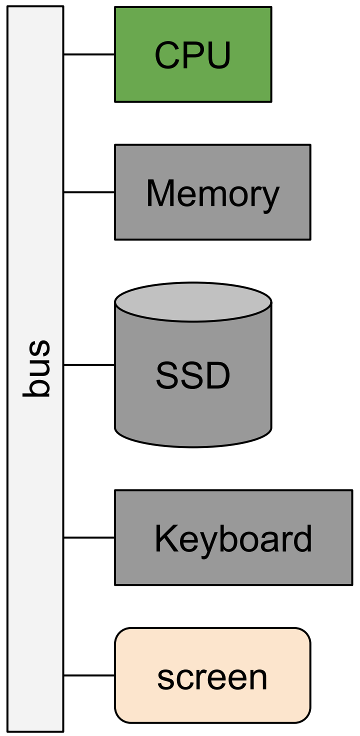
Reminder: write these questions in your notebook during lab

2. Variables and Literal Expressions

The Development Cycle



Review: Computer Architecture



A **program:**

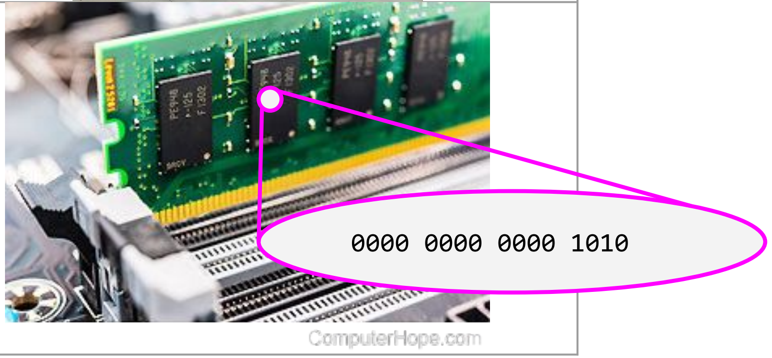
- Is made up of CPU instructions
- Tells the CPU to perform calculations and move data between memory, SSD, keyboard, screen, etc.
- Corresponds to an algorithm
- INPUT from keyboard or SSD
- OUTPUT to screen or SSD

This is all that programs do!

Objects and 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

Objects and Variables

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

Syntax: Variable Declaration and Initialization

statement:

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

Examples:

```
int count{ 0 };
double temperature{ 98.6 };
std::string name{"Ada"};
```

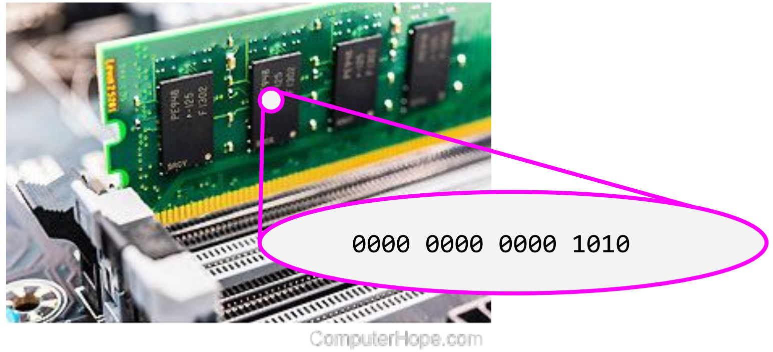
Semantics:

- 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**
  - Format for storing an object in memory
  - “Type” for short
- Will explore many data types
- For now, just 3:
  - `int`
  - `double`
  - `std::string`



```
int count{ 0 };
double temperature{ 98.6 };
std::string name{"Ada"};
```

Syntax: `int` Literal Expression

expression:

-(optional) *digit...*

Semantics:

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

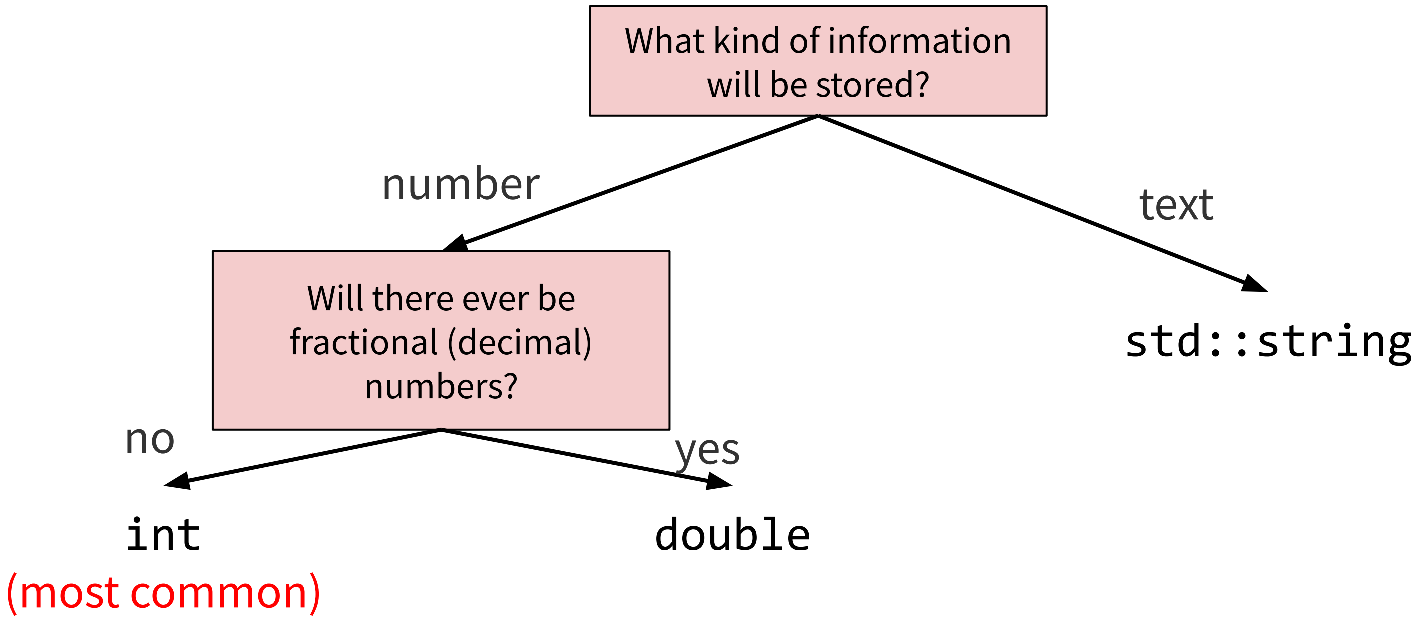
```
#include <iostream>

int main(int argc, char* argv[]) {
 int this_year{2022};
 int birth_year{1956};
 int age{this_year - birth_year};
 std::cout << "Age is " << age << "\n";

 return 0;
}
```

literal integers

`int`, `double`, `std::string`

Data Type	Kind of Information	Example Literal Value	Least Value	Greatest Value
`int`	integer (whole number)	120	-2.1 billion	2.1 billion
`double`	floating point number (decimal)	123.456	-1.8×10^{308}	1.8×10^{308}
`std::string`	text (words)	“Enter choice:”	“” (empty string)	only limited by computer memory

Choosing Between `int`, `double`, `std::string`



What kind of information will be stored?
number
text
std::string
Will there ever be fractional (decimal) numbers?
no
yes
int
(most common)
double

Identifier Rules

- Identifier may include
  - lower-case letter
  - upper-case letter
  - underscore _
  - digits (except first character)
- Must be at least one character
- Cannot be a keyword: `int`, `double`, `main`
- See Google C++ Style Guide: Variable Names

Valid identifiers:

`count`, `i`, `player_1`, `MAX_SCORE`

Invalid identifiers:

(empty), `count!`, `2player`

Recap: 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)`

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: `int` Literal Expression

expression:

-(optional) *digit...*

Semantics:

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

```cpp
#include <iostream>

int main(int argc, char* argv[]) {
 int this_year{2022};
 int birth_year{1956};
 int age{this_year - birth_year};
 std::cout << "Age is " << age << "\n";

 return 0;
}
```

literal integers

Syntax: `double` Literal Expression

literal double

expression:

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

Example:

```
double temperature{ 98.6 };
```

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*

```
#include <iostream>

int main(int argc, char* argv[]) {
 int this_year{2022};
 int birth_year{1956};
 int age{this_year - birth_year};
 std::cout << "Age is " << age << "\n";

 return 0;
}
```

Pitfall: Uninitialized Variables

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

Example: Undefined Behavior

```
#include <iostream>

int main(int argc, char* argv[]) {
 int year;
 std::cout << "Year is " << year << "\n";

 return 0;
}
```

All of these are possible outputs:

```
Year is 0
```

```
Year is -80401
```

```
Year is 2147483647
```

3. Input/Output

Standard Input/Output

- **Standard output**: text printed by program
- **Standard input**: text typed into program
- **`std::cout`**: standard output object
  - “c” for character
  - pronounced “see-out”
- **`std::cin`**: standard input object
  - Pronounced “see-in”

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 value is printed to standard output
  - Result type must be printable; otherwise compile error
- `int`, `double`, and `std::string` are all printable

Examples:

```
std::cout << 7

std::cout << “Hello” << “ there”

std::cout << (2 * 10)
```

Review: Pattern for Main Function Definition

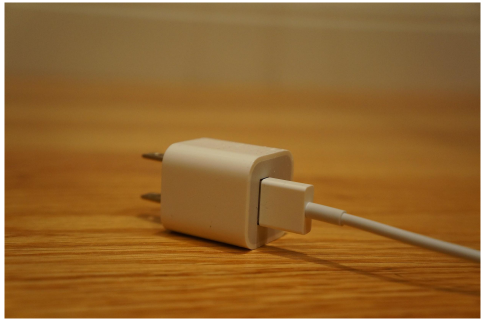
definition:

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: std::cout

```
#include <iostream>

int main(int argc, char* argv[]) {
 int year{2024};
 std::cout << "Year is " << year << "\n";

 return 0;
}
```

Output:

```
Year is 2024
```

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

```cpp
#include <iostream>

int main(int argc, char* argv[]) {
 int birth_year{0};
 int this_year{0};
 std::cout << "Enter birth year: ";
 std::cin >> birth_year;
 std::cout << "Enter this year: ";
 std::cin >> this_year;
 std::cout << "In " << this_year << ", a person"
           << " born in " << birth_year
           << " is " << (this_year - birth_year)
           << " years old\n";
 return 0;
}
```

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