Unit 1a – Basic Output and Input (with 'cout' and 'cin')

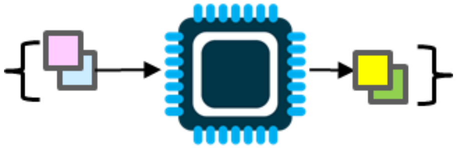
Mark Redekopp

Unit Objectives

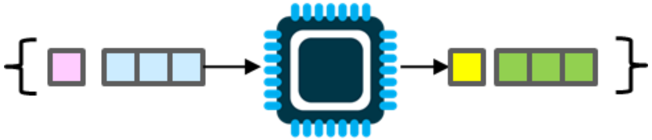
- List the various C data types
- Identify what type a constant is
- Know how to write constants in the appropriate C++ syntax
- Know the C++ operators and their order of operations
- Write basic output statements of text and constants using cout
- Use cin statement to get keyboard input from the user
- Predict how cin will treat input with whitespaces and extract data
- Know how variables are declared and assigned
- Trace the operation of assignment statements, expressions, and cin and cout commands

Unit

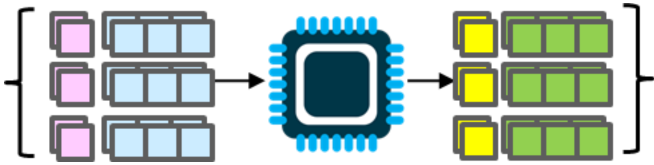
- **Unit 1**: Scalar processing
  - aka IPO=Input-Process-Output Programs



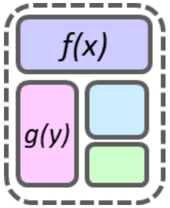
- **Unit 2**: Linear (1D) Processing



- **Unit 3**: Multidimensional Processing



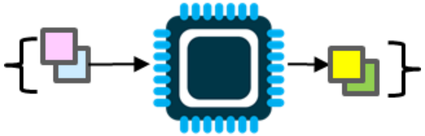
- **Unit 4**: Divide & Conquer (Functional Decomposition)

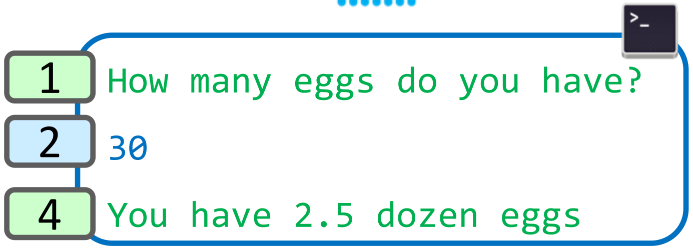


UNIT 1: SCALAR PROCESSING

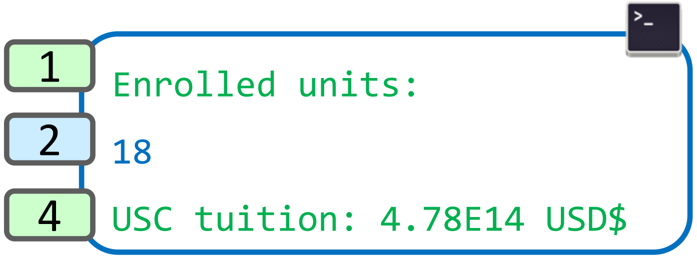
Scalar Processing Programs

- Scalar processing programs follow a simple sequence: Prompt-Input-Process-Output (PIPO)
  1. Prompt the user for **1 or more** (**some constant amount of**) input values
  2. Receive the input value(s)
  3. Using the input, perform operations (processing) to produce **1 or more** (**some constant amount of**) desired output values
  4. Display the output value(s)

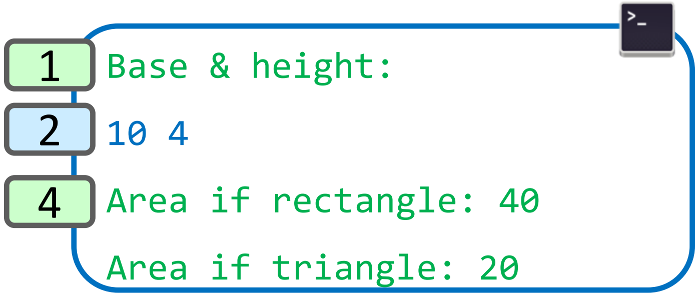




```
1 How many eggs do you have?
2 30
4 You have 2.5 dozen eggs
```



```
1 Enrolled units:
2 18
4 USC tuition: 4.78E14 USD$
```



```
1 Base & height:
2 10 4
4 Area if rectangle: 40
  Area if triangle: 20
```

Examples of Input-Process-Outptut programs.

Where is the processing?

PROCESSING

Basic Processing

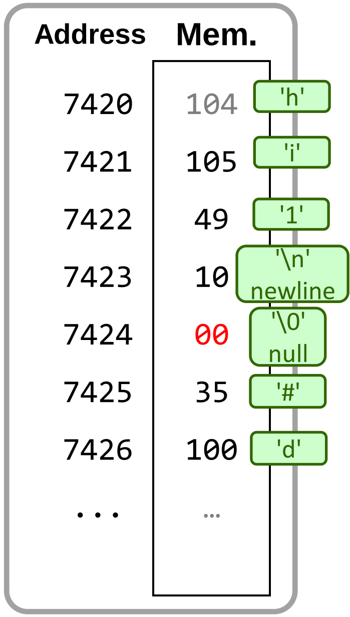
- To start, we will write programs to do simple processing similar to what we would use a calculator to perform.
- Let us briefly review the constants we introduced you to in an earlier unit.

Review: Constants (aka Literals)

- Integer: `496, 10005, -234`
- Double: `12.0, -16., 0.23, 6.02E23, 4e-2`
  - Both very large and very small numbers (i.e. fractions/decimals)
- Characters (char type): enclosed in **single quotes (')**
  - Printing characters: `'a', '5', 'B', '!'`
  - Each quoted value is converted to appropriate ASCII number (e.g. 'a' => 97)
  - Non-printing special characters use "escape" sequences (i.e. preceded by a \ ): `'\n'` (newline/enter), `'\t'` (tab) , `'\\'` (slash), `'\''` (apostrophe)
- C-Strings (Note: there is also a C++ string type…)
  - **0 or more** characters between **double quotes (")**

    `"hi1\n", "12345", "b", "\tAns. is %d"`
  - Ends with a `'\0'`=0 (aka NULL character) added as the last byte/character to allow code to delimit the end of the string
- Boolean (C++ only): `false, true`
  - Physical representation: 0 = false, Non-zero (1, -5, 300) = true

C/C++ handling of single characters and strings is different than most other languages and a major source of confusion in C++.



Address	Mem.	
7420	104	'h'
7421	105	'i'
7422	49	'1'
7423	10	'\n' newline
7424	00	'\0' null
7425	35	'#'
7426	100	'd'
...	…	

C-String Example (Memory Layout)

You're Just My Type

- Indicate which constants are matched with the correct type.

Constant	Type	Right / Wrong
4.0	int	
5	int	
'a'	string	
"abc"	string	
5.	double	
5	char	
"5.0"	double	
'5'	int	

Solutions are provided at the end of the slide packet.

Arithmetic Operators

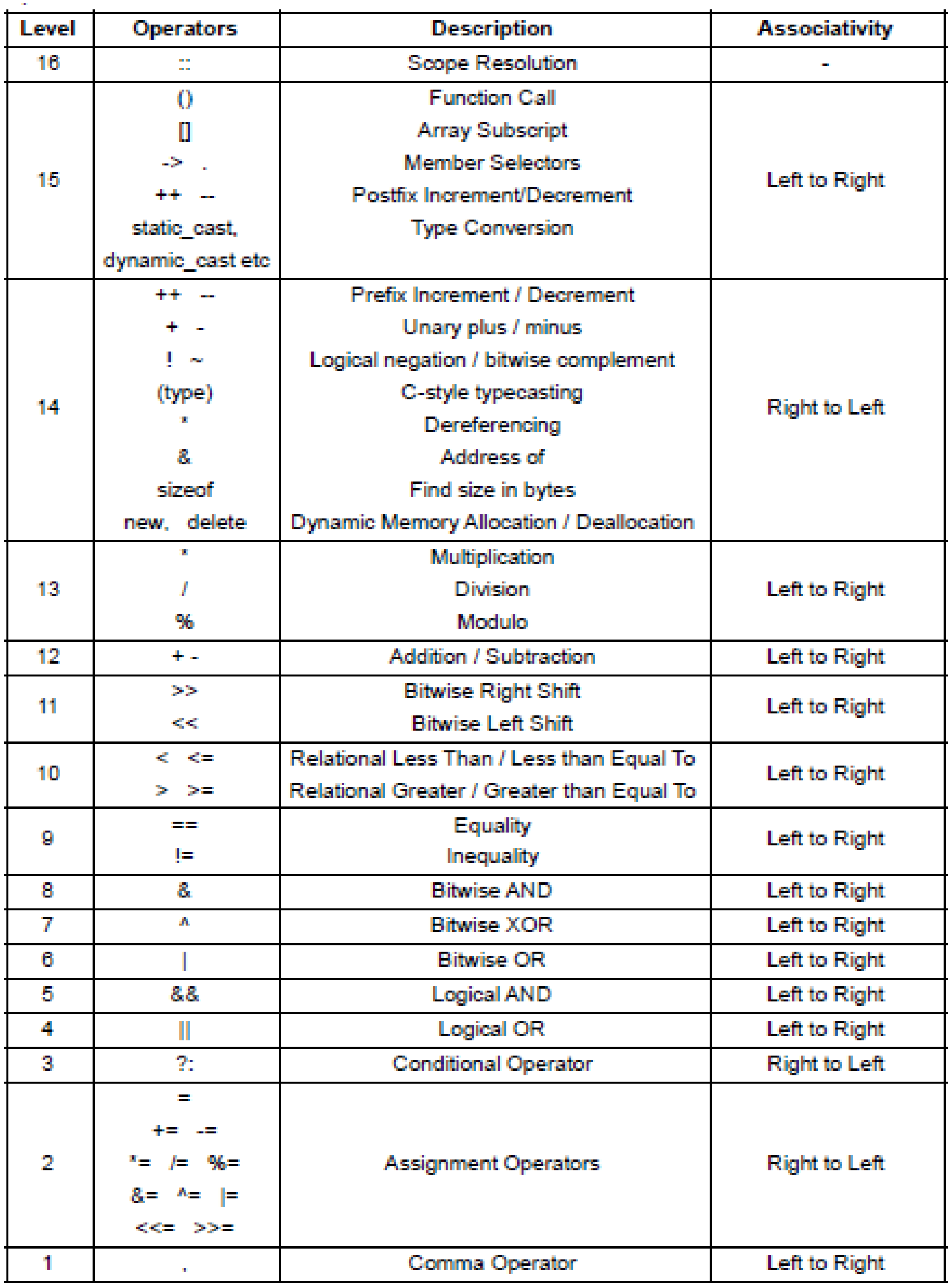
- Addition, subtraction, multiplication work as expected for both integer and floating point types
- Modulus is only defined for integers

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division (Integer vs. Double division)
%	Modulus (remainder) [for integers only]

```
10 % 3 = __
17 % 10 = __
```

Precedence

- Order of operations/ evaluation of an expression
- Higher level (level 16 in table) done first
- Notice operations with the same level or precedence usually are evaluated left to right)
- Evaluate:
  - `2*-4-3+5/2;`
- Tips:
  - Use parenthesis to add clarity
  - Add a space between literals `(2 * -4) – 3 + (5 / 2)`



Level	Operators	Description	Associativity
16	::	Scope Resolution	-
15	() [] -> . ++ -- static_cast, dynamic_cast etc	Function Call Array Subscript Member Selectors Postfix Increment/Decrement Type Conversion	Left to Right
14	++ -- + - ! ~ (type) * & sizeof new, delete	Prefix Increment / Decrement Unary plus / minus Logical negation / bitwise complement C-style typecasting Dereferencing Address of Find size in bytes Dynamic Memory Allocation / Deallocation	Right to Left
13	* / %	Multiplication Division Modulo	Left to Right
12	+ -	Addition / Subtraction	Left to Right
11	>> <<	Bitwise Right Shift Bitwise Left Shift	Left to Right
10	< <= > >=	Relational Less Than / Less than Equal To Relational Greater / Greater than Equal To	Left to Right
9	== !=	Equality Inequality	Left to Right
8	&	Bitwise AND	Left to Right
7	^	Bitwise XOR	Left to Right
6	|	Bitwise OR	Left to Right
5	&&	Logical AND	Left to Right
4	||	Logical OR	Left to Right
3	?:	Conditional Operator	Right to Left
2	= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operators	Right to Left
1	,	Comma Operator	Left to Right

https://discuss.codechef.com/upfiles/CPP.PNG

Review: Division

- Computers perform division differently based on the type of values used as inputs
- **Integer** Division:
  - When dividing two integral values, the result will also be an integer (any remainder/fraction will be dropped)
  - 10 / 4 = 2 52 / 10 = 5 6 / 7 = 0
- **Floating-point** (Double) & Mixed Division
  - 10.0 / 4.0 = 2.5 52.0 / 10 = 5.2 6 / 7.0 = 0.8571
  - Note: If one input is a double, the other will be promoted temporarily to compute the result as a double

Exercise Review

- Evaluate the following:

```
25 / 3
20 - 12 / 4 * 2
3 - 15 % 7
18.0 / 4
28 - 5 / 2.0
```

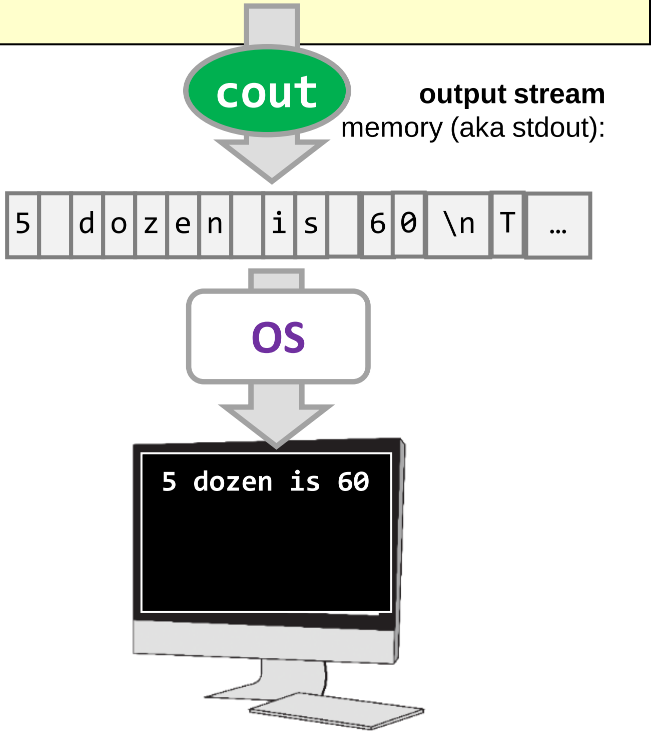
Exercises from: D.S. Malik, C++ Programming, 5th Ed., Ch. 2, Q6.

OUTPUT

C++ Output

- The most basic programming command is usually printing something to the screen
- In C++, this is done with a **cout** (short for character output) command
  - Different constants (and later variables) can be specified for output and cout will convert it all to text
  - The text is then handed to the OS to be printed
  - endl = '\n' (newline)

```
#include<iostream>
using namespace std;
int main()
{
  cout << "5 dozen is: " << 5*12 << endl;
  cout << "The end";
  return 0;
}
```



```
5 dozen is 60
```

This Photo by Unknown Author is licensed under CC BY-NC

Output From Your Program

- To see output in C++ we need to explicitly tell the computer to output the value using 'cout'
  - So what happens to the result of 12*3 on the first line?
- Note: 'endl' stands for **end-line** and causes the cursor to move to the next line of the screen similar to '\n'

Performing computation is like having a thought. No output is generated unless you explicitly write it down.

To output a result to the screen in C++ (i.e. "write it down") we use the 'cout' command

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  12 * 3;                    // No result printed

  cout << 12 * 3 << endl; // 36 printed

  return 0;
}
```

This Photo by Unknown Author is licensed under CC BY-SA-NC
This Photo by Unknown Author is licensed under CC BY-SA-NC

Printing Different Values & Types

- 'cout' requires appropriate use of **separators** between consecutive values or different types of values
- 'cout' does not add spaces between consecutive values; you must do so explicitly
  - Since text strings are a different value we must separate it with the '<<' operator
- Generally good practice to give some descriptive text with your numeric output
  - Note: You may divide up output over multiple 'cout' statements. Unless an 'endl' or '\n' is used, the next 'cout' statement will resume where the last one left off

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

int main()
{
  cout << 102 1889 << endl;    // Compile Error!
  cout << 102 << 1889 << endl; // Better, but no spaces
  cout << 102 << " " << 1889 << endl; // Best
  cout << "102 1889" << endl;  // or as a string

  cout << "There are " << 60*24*365 << " ";
  cout << "minutes in a year." << endl;
  return 0;
}
```

```
1021889
102 1889
102 1889
There are 525600 minutes in a year.
```

The << operator has multiple (aka "overloaded") meanings. In C (and still in C++) it is used to shift bits in a variable to the left, but C++ also uses it for output. In that (output) context, it is NOT known as the shift operator but the "stream insertion" operator!

Challenge 1

- Write a program that incorporates:
  - Processing
  - Output
- Think of simple converters/calculator operations to work with a fixed input
  - Example: How many hours will it take to drive 850 **miles** at 110 **km/h**?

VARIABLES AND RECEIVING INPUT WITH CIN

The Need For Variables & Input

- Printing out constants is not very useful (nor exciting)
- In fact, we could just as easily compute the value ourselves in many situations
- The real power of computation comes when we introduce **variables** and user input via **`cin`**
  - **Variables** provide the ability to remember and name a value for use at a later time
  - **User input** allows us to write general programs that work for "any" input values
  - Thus, a more powerful program would allow us to enter an arbitrary number and perform conversion to dozens

```
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  cout << "3 dozen is " << 3*12 << " items." << endl;

  // the above results in the same output as below

  cout << "3 dozen is 36 items." << endl;

  return 0;
}
```

```
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  int dozen;
  cout << "How many dozen do you have: " << endl;
  cin >> dozen;
  cout << "You have" << 12*dozen << " items." << endl;
  return 0;
}
```

C/C++ Variables

- A variable is a reserved memory location that
  - **Stores a value that can be read (retrieved) or written (changed) as often as desired**
  - **Associates a descriptive name (e.g. x) the programmer will use with that memory location (aka address) and the value stored in that location**
- **It's like renting an apartment or storage unit**
- You must "**declare**" (allocate) your variables before using/assigning to them
  - A variable is not allocated (aka "in scope") until the computer executes the line with the declaration
  - The declaration must give the **type** of the variable and a **name/identifier**

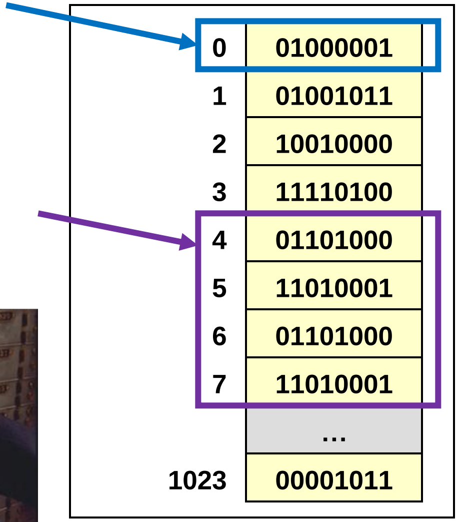
```
#include <iostream>
using namespace std;

int main()
{ // Sample variable declarations
  gr = 'A'; // BAD! must declare first
  char gr = 'A'; // GOOD! Declared 'gr'
  int x;   // uninitialized variables
           // will have a (random) garbage
           // value until we initialize it
  x = 1;   // Initialize x's value to 1
  gr = 'B'; // Change gr's value to 'B'
}
```

Difference: C required that variables be declared at the beginning of a function before any operations.
C++ relaxes this and allows declarations anywhere in the code.

A picture of computer memory (aka RAM)

char gr = 'B';
A single-byte variable

int x;
A four-byte variable



0	01000001
1	01001011
2	10010000
3	11110100
4	01101000
5	11010001
6	01101000
7	11010001
	…
1023	00001011

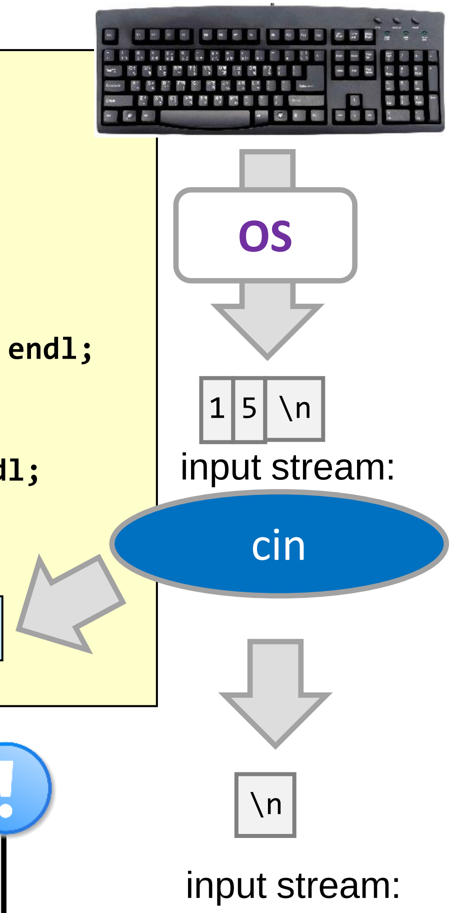
Variables are actually allocated in RAM when the program is run

Keyboard Input

- In C++, the 'cin' object is responsible for receiving input from the keyboard
- Keyboard input is captured and stored by the OS (in an "input stream") until cin is called upon to "extract" info into a variable in your program
- 'cin' converts text input to desired format (e.g. integer, double, etc.)

```
#include <iostream>
using namespace std;

int main()
{
  int dozens;

  cout << "Enter number of dozen: " << endl;
  cin  >> dozens;

  cout << 12 * dozens << " eggs" << endl;
  return 0;
}
```

dozens 15



> The >> operator also has multiple (aka "**overloaded**") meanings. In C (and still in C++) it is used to **shift** bits in a variable to the right, but C++ also uses it for input. In that (input) context, it is known not as the shift operator but the "**stream extraction**" operator!

Dealing With Whitespace

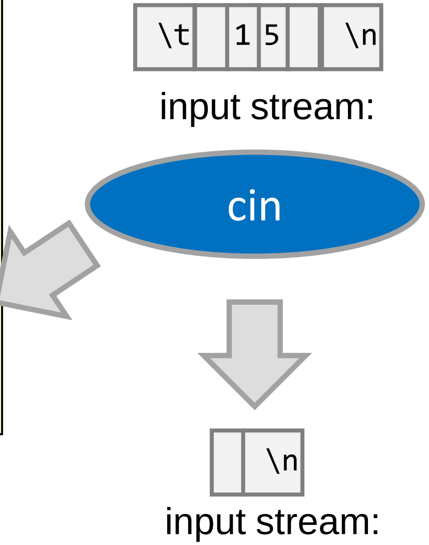
- ***Whitespace (def.):***
  - Characters that represent horizontal or vertical blank space. Examples: newline ('\n'), TAB ('\t'), spacebar (' ')
- cin sequentially discards **leading** whitespace characters until it hits a non-whitespace.
- cin then checks the characters can be converted to the appropriate variable type and keeps scanning for more
- cin will STOP at the first **trailing** whitespace (or on a character unable to be converted to the desired type) and await the next cin command

```
#include <iostream>
using namespace std;

int main()
{
  int dozens;

  cout << "Enter number of dozen: "
       << endl;
  cin  >> dozens;

  cout << dozens << " dozen "
       << " is " << 12*dozens
       << "items." << endl;

  return 0;
}
```

15

dozens

Suppose at the prompt the user types:



Main Take-aways:

cin SKIPS leading whitespace

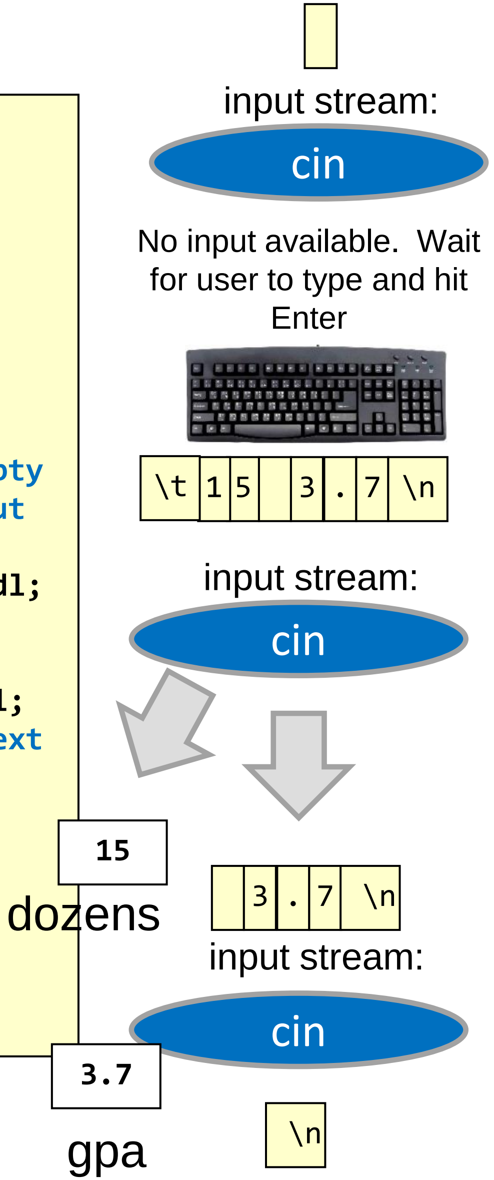
cin STOPS on the first trailing whitespace

Space ≠ Whitespace (Whitespace = ' ', '\t', '\n', etc.)

Timing of Execution

- When execution reaches a 'cin' statement, it will either:
  - **Wait** for input if nothing is available in the input stream
    - OS will capture what is typed until the next 'Enter' key is hit
    - User can type as little or much as desired until Enter (\n)
  - **Immediately extract** from the input stream if some text is available and convert it to the desired type of data

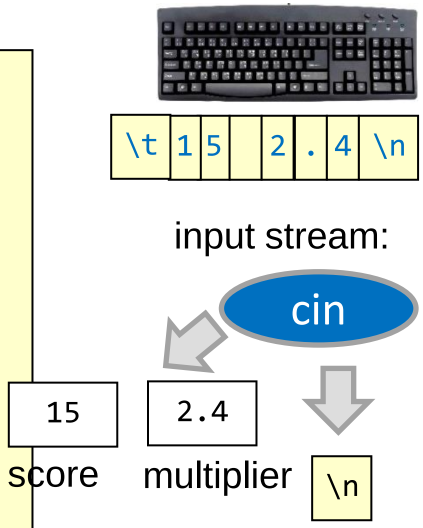
```
#include <iostream>
using namespace std;

int main()
{
  int dozens;

  cout << "Enter number of dozen: "
       << endl;
  cin  >> dozens; // input stream empty
                  // so wait for input

  cout << 12*dozens << " eggs" << endl;

  double gpa;
  cout << "What is your gpa?" << endl;
  cin  >> gpa; // input stream has text
               // so do not wait…
               // just use next text

  cout << "GPA = " << gpa << endl;
  return 0;
}
```

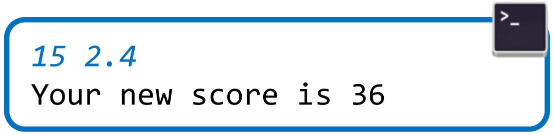

No input available. Wait for user to type and hit Enter

Multiple Inputs and Unexpected Inputs

- Use the '>>' operator to separate any number of variables you want to read
- For now let us make the unreasonable assumption that the user always types in the write "format" of information
  - We'll learn more about how cin handles errors later

```
#include <iostream>
using namespace std;

int main()
{
  int score;
  double multiplier;
  cin >> score >> multiplier;

  cout << "Your new score is "
       << score * multiplier << endl;

  return 0;
}
```





```
15 2.4
Your new score is 36
```

Challenge 2

- Write a program that incorporates all 3 aspects:
  - Input
  - Processing
  - Output
- Example: Compute and output how far an object with initial downward velocity, **v**, has fallen after **t** seconds?

SOLUTIONS

You're Just My Type

- Indicate which constants are matched with the correct type.

Constant	Type	Right / Wrong
4.0	int	double (.0)
5	int	int
'a'	string	char
"abc"	string	C-string
5.	double	float/double (. = non-integer)
5	char	Int…but if you store 5 in a char variable it'd be okay (char = some number that fits in 8-bits/1-byte
"5.0"	double	C-string
'5'	int	char

Exercise Review

- Evaluate the following:
  - `25 / 3 = 8`
  - `20 - 12 / 4 * 2 = 14`
  - `3 - 15 % 7 = 2`
  - `18.0 / 4 = 2.5`
  - `28 - 5 / 2.0 = 25.5`

Exercises from: D.S. Malik, C++ Programming, 5th Ed., Ch. 2, Q6.