PIC 10A: Week 3b

Section 1C, Winter 2016

Prof. Michael Lindstrom (TA: Eric Kim)

v1.0

Announcements

- HW2 is now due *tonight* at 11 PM
- Several HW1 scores were updated
    - Can check score+comments on ccle
-

Today

- Operators, shorthands (i++, j+=3, etc.)
  - Mod operator: %
- Numerical errors
- cin input buffering
- Discussion Problems

Operator Shorthands

```
int i = 0;
```

$$\underbrace{\texttt{i++;} \qquad\qquad \texttt{++i;} \qquad\qquad \texttt{i += 1;}}_{}$$

All have the same effect:
increment i by 1.

Also equivalent: `i = i + 1;`

The Mod Operator: %

- AKA remainder
- Syntax: a % b, where a,b are **positive** integers.
  - In this course: Never let a or b be negative
- In math notation, we express this as:
  - a mod b

```
int x = 1 % 3;    // x is 1
x = 2 % 3;    // x is 2
x = 3 % 3;    // x is 0
x = 4 % 3;    // x is 1
x = 5 % 3;    // x is 2
x = 6 % 3;    // x is 0
...
```

Trivia: According to C++ spec: if either are negative, then results are "implementation-defined", which means different compilers/machines are allowed to give whatever result they like. Scary.

Mod

Think of `5 % 3` as: the *remainder* of doing 5/3:

$$5/3 = 1 + (\underbrace{\mathbf{2}}/3)$$

`5 / 3 =` **`1`**

Integer division yields the **quotient**.

`5 % 3 =` **`2`**

The **remainder**

Numerical Errors

- Overflow: When the value of an int/double exceeds the maximum value
    - Example: Recall that an int has a range of about -2 billion to +2 billion.

```
int x = 2e9; // 2 billion
cout << "x=" << x << endl;
cout << "2*x=" << 2*x << endl;
```

Output:

```
x=2000000000
2*x=-294967296
```

$\underbrace{}_{\text{Woah, is negative?!}}$

Numerical Errors

- **Underflow** is when a value is smaller than the data type's smallest value
- Precision errors
  - Recall: double has roughly 15 digits of precision

```
double a = 2;
double b = sqrt(a)*sqrt(a) - 2;
cout << "sqrt(2)*sqrt(2) - 2 = " << b << endl;
```

Output:

```
sqrt(2)*sqrt(2) - 2 = 4.44089e-016
```

Woah, not exactly 0!

cin: Input buffering

```
int x;
cin >> x;
```

What cin does:
(1) **Skip** all whitespace (spaces, tabs, newlines) until it reaches a non-whitespace character.
(2) Attempts to interpret the current character as the desired type (int, double, string, etc.).
    If <u>success</u>: **chomp** the character, and move onto the next character. Stops as soon as we either find whitespace, or an inappropriate character.
    If <u>fail</u>: cin issues a failure, and **"passes out"** until you fix it.

cin: Example

```
int x;
double y;
cin >> x;
cin >> y;
```

Suppose the user typed:

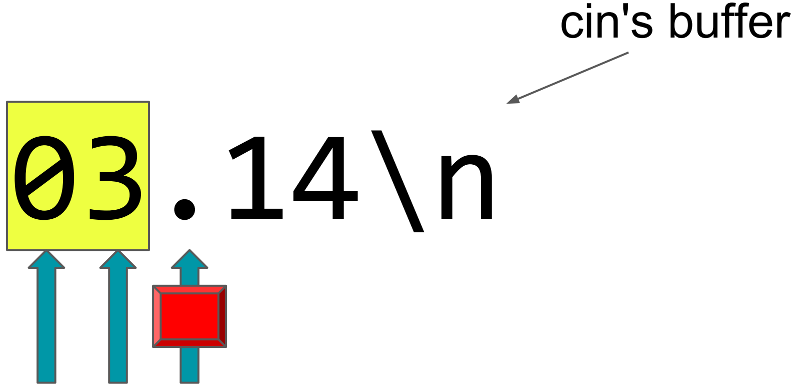
```
03.14<ENTER>
```

What're the values of x and y?

Answer: x is 3, and y is 0.14

cin: Step by Step

```
int x;
double y;
cin >> x;
cin >> y;
```



'.' is not valid for type `int`!

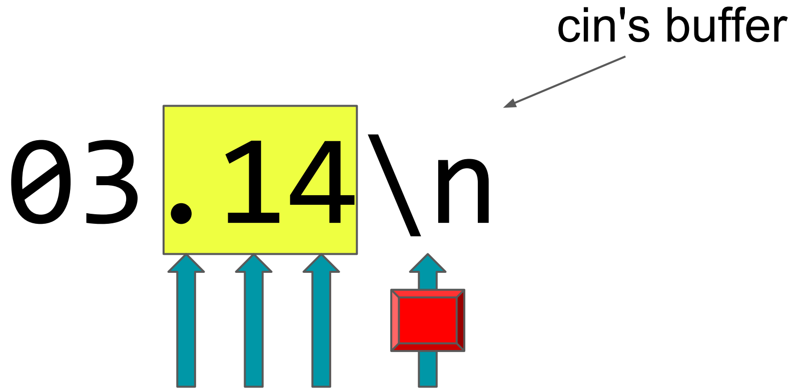
User typed:
`03.14<ENTER>`

Outcome:
cin sets x to: 3

Note: cin remembers its position in the buffer for next time.

cin: Step by Step

```
int x;
double y;
cin >> x;
cin >> y;
```



Stop: reached a whitespace character (newline).

User typed:
`03.14<ENTER>`

Outcome:
cin sets y to: 0.14

cin: Handling Errors

```
int x, y;
cin >> x;
cout << "x is: " << x << endl;;
cin >> y;
cout << "y is: " << y;
```

Suppose user types:

```
d3<ENTER>
```

Output:

```
x is: -858993460
y is: -858993460
```

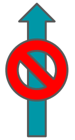
Uhoh! x, y not set.
Note: we are using x **without initializing** it with a value, hence why this value is so strange.

cin: Handling Errors

```
int x,y;
cin >> x;
cin >> y;
```

cin's buffer

```
d3\n
```



cin status: **FAILURE**.

cin sees that 'd' is **invalid** for type `int`.

User typed:
`d3<ENTER>`

Outcome:
cin enters a failure state, and "passes out".
cin does not set x to any value.
Any further attempts to use cin will **not do anything**!

cin: Handling Errors

```
int x;
int y;
cin >> x;
cin >> y;
```

cin's buffer

```
d3\n
```

cin status: **FAILURE**.

cin is in a failure state, so **does nothing**.

User typed:
`d3<ENTER>`

Outcome:
cin does not set y to any value.

cin: How to fix failure state?

`cin.clear()` is a function that resets cin's state from "Failure" to "Good".

Use it to wake up a "passed out" cin.

cin.clear(): Example

```
int x = 8;
double y;
cin >> x;
cin.clear();
cin >> y;
cout << "x: " << x << endl;
cout << "y: " << y;
```

User types:

```
.45<ENTER>
```

Question: What is the output?

Output:

```
x: 8
y: 0.45
```