cin and Arithmetic

Class 7

From Lab 2

- some things noticed in lab 2 submissions

```
double diameter;
diameter = radius * 2.0;
```

```
double diameter = radius * 2.0;
```

Which is preferable, and why?

From Lab 2

- some things noticed in lab 2 submissions

```
double diameter;
diameter = radius * 2.0;
```

```
double diameter = radius * 2.0;
```

Which is preferable, and why?

The right side, because it's more concise. It conveys the exact same meaning with fewer words, and is thus more readable.

From Lab 2

- some things noticed in lab 2 submissions

```
double diameter;
diameter = radius * 2.0;
```

```
double diameter = radius * 2.0;
```

Which is preferable, and why?

The right side, because it's more concise. It conveys the exact same meaning with fewer words, and is thus more readable.

```
diameter
radius
area
```

```
circle_diameter
circle_radius
circle_area
```

Which is preferable, and why?

From Lab 2

- some things noticed in lab 2 submissions

```
double diameter;
diameter = radius * 2.0;
```

```
double diameter = radius * 2.0;
```

Which is preferable, and why?

The right side, because it's more concise. It conveys the exact same meaning with fewer words, and is thus more readable.

```
diameter
radius
area
```

```
circle_diameter
circle_radius
circle_area
```

Which is preferable, and why?

The left side, because this program only involves a circle. If this were a general-purpose geometry program, then triangle_area vs circle_area might be essential. But here, circle_ is simply noise.

From Lab 2

```
const double PI_CONSTANT = 3.1415;
        or
const double MATHEMATICAL_PIE = 3.1415;
        or
const double PI = 3.1415;
```

Which one is preferable, and why?

From Lab 2 (and Lab 3)

`diam`	`diameter`
`rad`	`radius`
`circ`	`circumference`

Which is preferable, and why?

From Lab 2 (and Lab 3)

`diam`	`diameter`
`rad`	`radius`
`circ`	`circumference`

Which is preferable, and why?

The left side is unacceptable. There is simply no excuse for this kind of abbreviation. diam might be barely unambiguous, but rad and circ are too likely to be misunderstood (radians, radical, radio, or radicchio?). Do not get into the habit of abbreviating like this.

From Lab 2 (and Lab 3)

`diam`	`diameter`
`rad`	`radius`
`circ`	`circumference`

Which is preferable, and why?

The left side is unacceptable. There is simply no excuse for this kind of abbreviation. diam might be barely unambiguous, but rad and circ are too likely to be misunderstood (radians, radical, radio, or radicchio?). Do not get into the habit of abbreviating like this.

In lab 3, I hope you aren't making variables named C or Y!

From Lab 2

From the assignment: " ... the program should produce output that looks exactly like this:"

```
Please enter the radius of a circle: 14
For a circle with radius 14
The diameter is 28
The circumference is 87.962
The area is 615.734
```

From a student's program:

```
This program Calculates the properties of a circle
given its radius
Please enter the circle's radius: 14
The diameter is 28 units
The circumference is 87.962 units
The area is 615.734 units
```

From Lab 2

From the assignment: " ... the program should produce output that looks exactly like this:"

```
Please enter the radius of a circle: 14
For a circle with radius 14
The diameter is 28
The circumference is 87.962
The area is 615.734
```

From a student's program:

```
This program Calculates the properties of a circle
given its radius
Please enter the circle's radius: 14
The diameter is 28 units
The circumference is 87.962 units
The area is 615.734 units
```

If the specifications call for a particular behavior, you the developer must meet those specifications.

From Lab 3

From the assignment: " ... the program should produce output that looks exactly like this:"

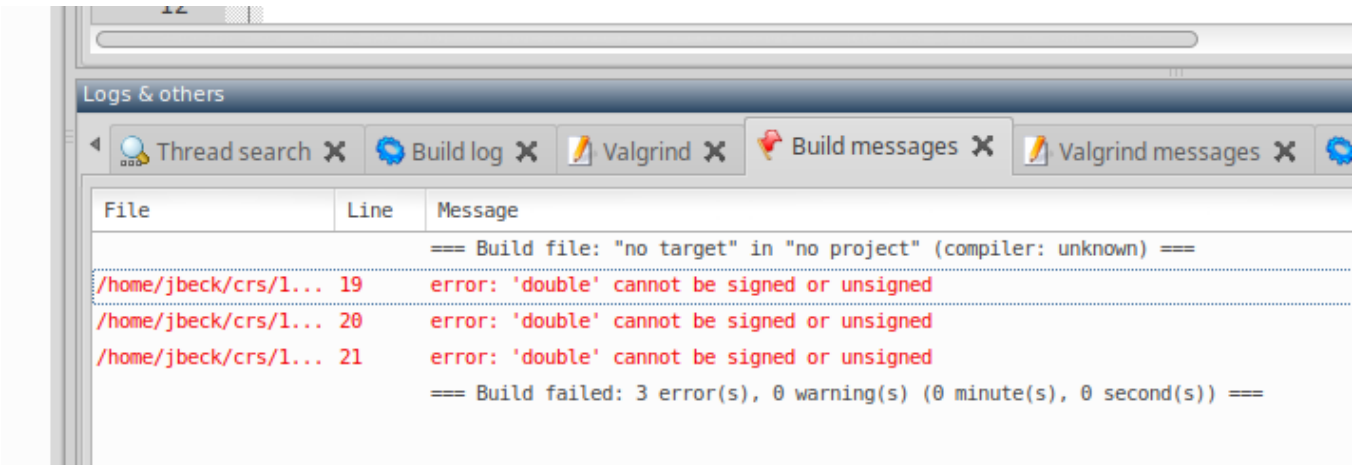
```
Enter the number of completions: 329
Enter the number of attempts: 483
Enter the number of yards: 4024
Enter the number of touchdowns: 34
Enter the number of interceptions: 8

The player's rating is: 110.123
```

make sure your program produces a screen that looks exactly like this when it runs

From Lab 2

- if you get any build warnings or errors, you will lose points
- their absence doesn't mean your program is correct
- but their presence definitely means something is wrong



The cin Object

- cout is an output stream object connected to the screen
- cin is an input stream object connected to the keyboard
- cout is used with the stream insertion operator <<
- cin is used with the stream extraction operator >>

- the stream extraction operator extracts a value from the stream of characters coming from the keyboard and puts that value into the operator's rhs variable

The cin Object

- the variable that is used with cin must have been declared prior to use
- you cannot initialize a variable with cin

OK:

```
double radius;
cin >> radius;
```

Illegal:

```
cin >> double radius;
```

cin with Multiple Values

- cin extraction may be used to gather multiple values in one statement
- see program 3-2 on page 87:

```
unsigned length;
unsigned width;
cout << "Enter the length and width: ";
cin >> length >> width;
unsigned area = length * width;
```

Multiple Values of Varying Types

- cin can extract multiple values of different types from the input stream

```
unsigned count;
double measurement;
cout << "Enter the count and the measurement: ";
cin >> count >> measurement;
```

Whitespace

- whitespace is the term used to describe any sequence of one or more characters that the eye perceives as a separation of words
- the main whitespace characters are space, tab, and newline

Keyboard Buffer

- in computer science, a buffer is an area of memory where characters are stored while waiting to be processed
- when you type on a keyboard, the keystrokes are stored in the keyboard buffer until a program processes them
- the program only starts reading the characters when the user presses the Enter key
- all characters before the Enter key are stored in the keyboard buffer
- you can backspace and change characters as long as the Enter key has not been pressed
- when the Enter key is pressed, all characters plus the Enter key are sent to the program

Delimiting cin Input

- cin extraction skips whitespace before starting to extract a value

- cin extraction is greedy as it reads the keyboard stream
- it tries to read as many characters as possible
- it stops reading only when it can't go farther

- cin extraction throws away everything in the buffer between the last thing it can read and the Enter key

run program try_cin with various inputs, correct and incorrect:

cin Notes

- cin is very easy to use, BUT:
  - it is impossible to read the space character into a char variable
  - cin extraction cannot read a string with an embedded space
  - cin extraction does no error checking
  - cin extraction can give incorrect results even when there is technically no error
- we will later see better and safer ways of getting input

Combined Assignment

- the most common statement in programming is assignment:
  `foo = bar;`
- the second most common statement pattern is an arithmetic operation on a variable followed by assignment to that variable:
  `foo = foo + 3;`
- because this pattern is so common
- because it involves the name of the variable typed twice
- C++ has a shortcut form that combines arithmetic and assignment in one symbol

```
foo = foo + 3;
foo += 3;
```

Combined Assignment

`foo += 3;`	retrieve the current value of foo, add three to it, and store the result in foo
`foo *= 3;`	retrieve the current value of foo, multiply it by three, and store the result in foo
`foo /= bar - 5;`	retrieve the current value of bar, subtract 5 from that value, use the result as the divisor of the current value of foo, and store the final result in foo; bar is not changed

Operator Precedence and Associativity

- when multiple operators exist in a single statement
- the order in which they are evaluated depends on a combination of precedence and associativity
- for the operators we have seen, the precedence and associativity are as follows

Operator	Precedence	Associativity
$-$ (unary)	1	left-to-right
* / %	2	left-to-right
$+$ $-$ (binary)	3	left-to-right
$=$ *$=$ $/=$ $+=$ $-=$ %$=$	4	right-to-left

Multiple Operators

- when multiple operators of different precedence exist in a single statement
- precedence determines order

```
foo = bar * bim - bam;
```

- in order of precedence, the operators are executed:
  1. * (highest precedence)
  2. − (medium precedence)
  3. = (lowest precedence)

Multiple Operators

- when multiple operators of the same precedence exist in a single statement
- associativity determines order

```
foo / bar * bim % bam
```

- an expression, not a complete statement
- these operators have equal precedence and left-to-right associativity
- they are executed:
  1. / (leftmost)
  2. * (middle)
  3. % (rightmost)

Multiple Operators

- when multiple operators of the same precedence exist in a single statement
- associativity determines order

```
foo = bar = bim = 5;
```

- these operators have equal precedence and right-to-left associativity
- they are executed:
  1. bim = 5 (rightmost)
  2. bar = result of 1 (middle)
  3. foo = result of 2 (leftmost)
- foo, bar, and bim all get the value 5

Algebra vs. Programming

- human algebra uses some syntax and shortcuts that are not available in C++

Algebraic Expression	**C++ Equivalent**
$6b$	`6 * b`
$(3)(12)$	`3 * 12`
$x = \frac{a+b}{c}$	`x = (a + b) / c;`
$y = 3\frac{x}{2}$	`y = 3 * (x / 2);`

Exponentiation

- C++ does not have an exponentiation operator
- the pow function is provided by the cmath library
- regardless of arguments, pow returns a double value

```
#include <cmath>
        ...
 foo = pow(bar, 2.5);
```

- for simple exponentiation like squaring, it's faster and easier to simply multiply the variable by itself
  `foo = bar * bar;`