

# Collections, Part One

# Announcements

- Section signups open today at 5PM and close Sunday at 5PM.
- Sign up for section at  
**<http://cs198.stanford.edu/section>**
- Link available on the CS106B course website.

# In Person vs. Remote Sections

- In order to keep section sizes small we are offering two types of sections
  - Regular in-person sections with local section leaders
  - “Tele-sections” via Google Hangouts (think skype with video) with section leaders who are not in the Stanford area.
- Section content is exactly the same and you can still ask questions during section.
- If you strongly prefer one over the other, then in your section preferences only select sections of the form you want.

`console.h, cout and endl`

- Some people running Windows have been having issues with the console window quickly disappearing
- It appears that for many people the issue can be solved doing one or both of the following:
  - Downloading the latest version Java
  - Passing `endl` to `cout` at least once in your program

# Where are we in the course?

- For the moment we are done with C++ specific features
- Today we start learning about common data structures used in Computer Science
- After this we have...
  - Advanced Recursion
  - Algorithmic Analysis and Sorting
  - Implementing data structures
  - Graphs and Graph Algorithms

# Organizing Data

- In order to model and solve problems, we have to have a way of representing structured data.
- We need ways of representing concepts like
  - sequences of elements,
  - sets of elements,
  - associations between elements,
  - etc.

# Collections

- A **collection class** (or **container class**) is a data type used to store and organize data in some form.
- Understanding and using collection classes is critical to good software engineering.
- Today and next week is dedicated to exploring different collections and how to harness them appropriately.
- We'll discuss efficiency issues and implementations later on.

# Collections

- There are TONS of C++ libraries for collection classes
  - General Purpose: STL, Boost
  - Most companies have their own libraries
- So which library should we teach you?
- Because there are so many libraries, we think it's best to focus on skills and concepts, rather than on one specific library.
- At Stanford, we decided to create our own library for CS106B which we've optimized to be easy to learn and use.



# TokenScanner

- The **TokenScanner** class can be used to break apart a string into smaller pieces.
- Construct a `TokenScanner` to piece apart a string as follows:

```
TokenScanner scanner (str) ;
```

- Configure options (ignore comments, ignore spaces, add operators, etc.)
- Use the following loop to read tokens one at a time:

```
while (scanner.hasMoreTokens ()) {  
    string token = scanner.nextToken ();  
    /* ... process token ... */  
}
```

- Check the documentation for more details; there are some really cool tricks you can do with the `TokenScanner`!

# Text Parsing

- TONS of websites that you can download data from in the form of “comma-separated-values” (csv) files.
  - e.g. Financial Data, Climate data
- Problem: Have a string consisting of a long sequence of numbers separated by commas.
- Goal: Extract numbers and calculate their average
- **How tough would this be using string libraries?**

ComputeSum.cpp  
(On Computer)

Stack

# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



# Stack

137

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls

42

137



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls





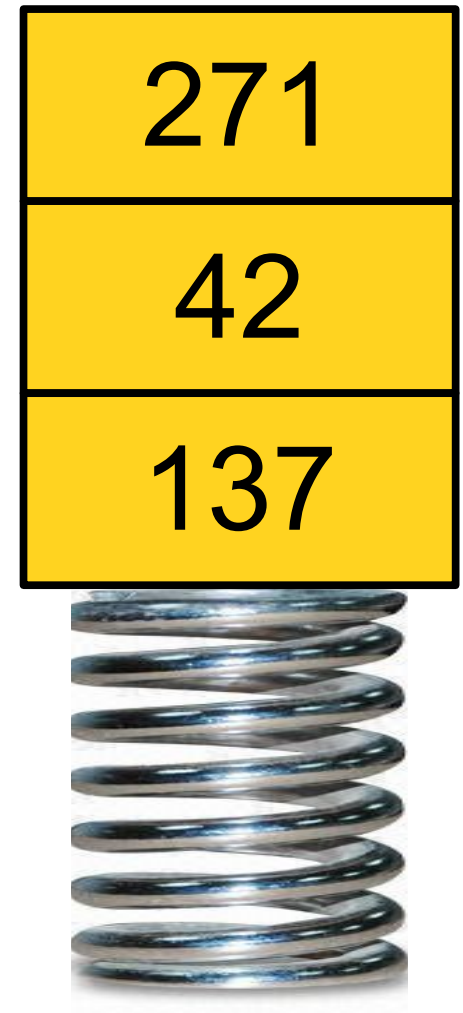
# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



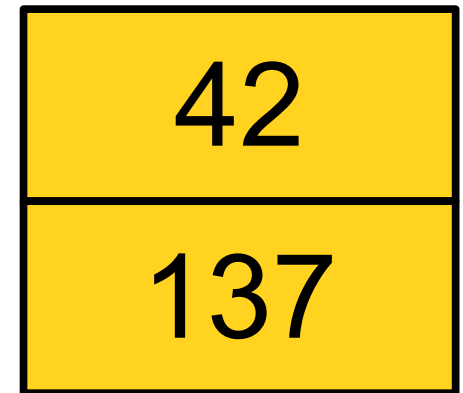
# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



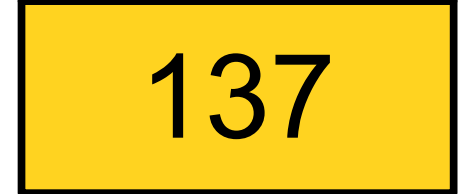
# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



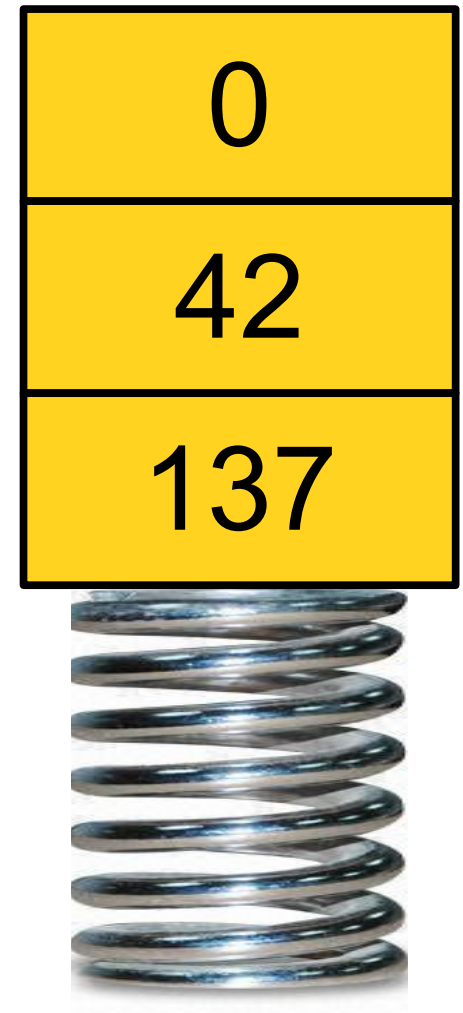
# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



# Stack

- A **Stack** is a data structure representing a stack of things.
- Objects can be **pushed** on top of the stack or **popped** from the top of the stack.
- Only the top of the stack can be accessed; no other objects in the stack are visible.
- Example: Function calls



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

^



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

^



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

^



# Balancing Parentheses

```
int foo(^) { if (x * (y + z[1]) < 137) { x = 1; } }
```





# Balancing Parentheses

```
int foo(^) { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



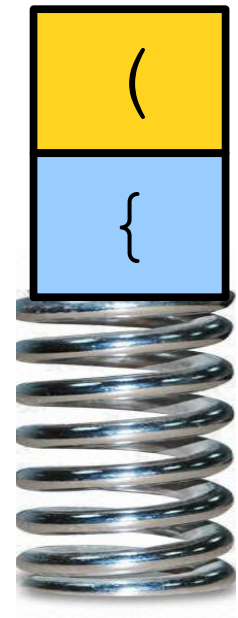
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



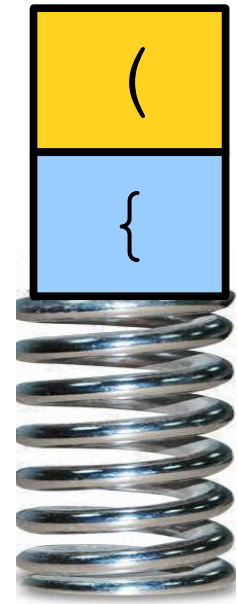
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



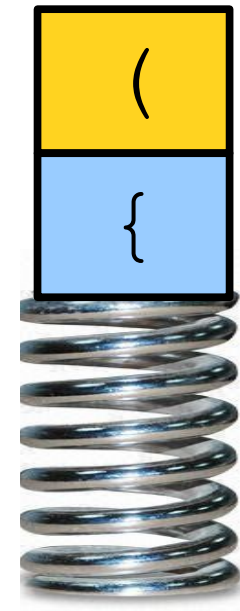
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



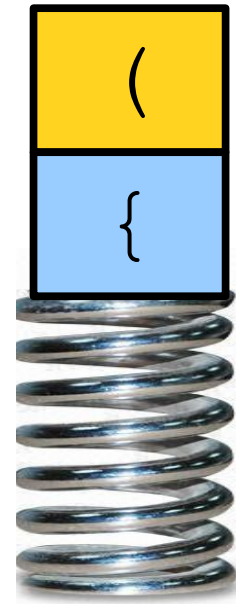
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



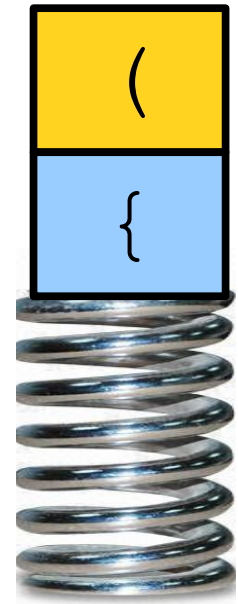
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

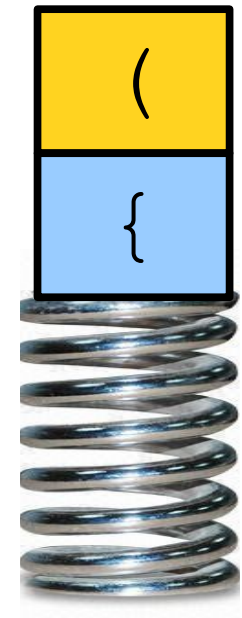
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





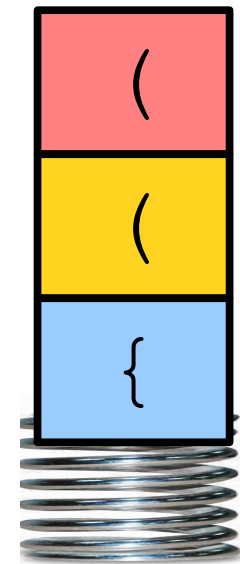
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



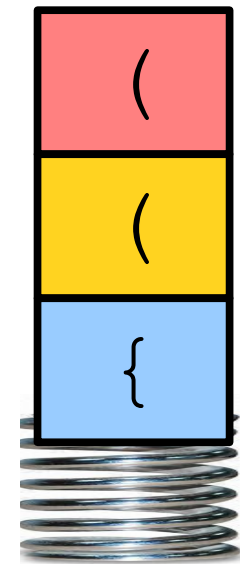
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



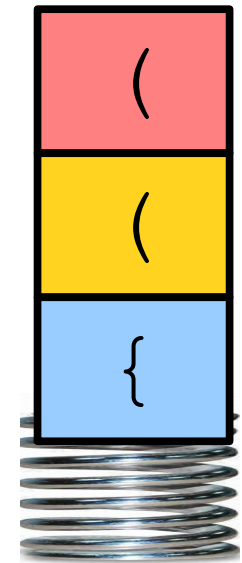
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



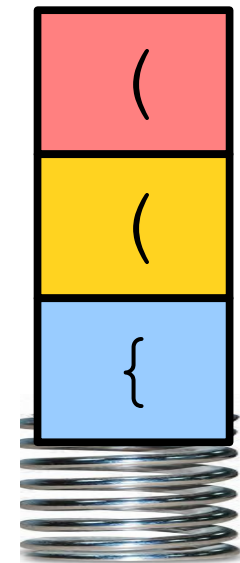
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



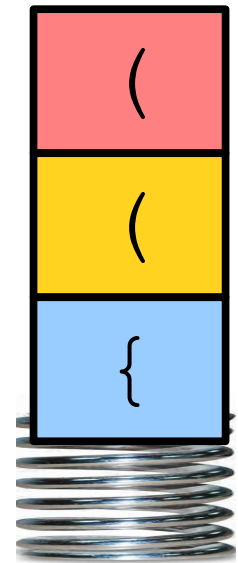
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



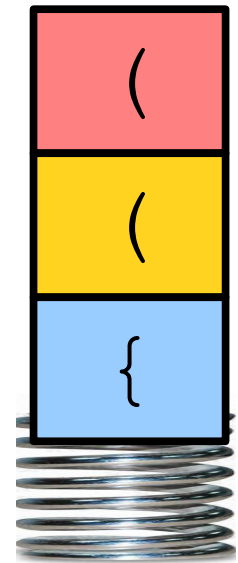
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



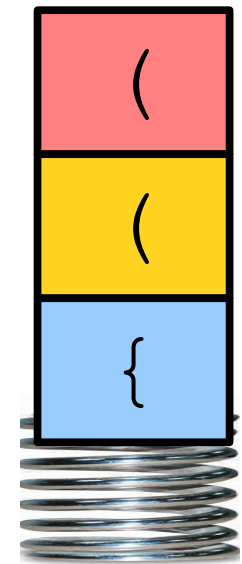
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

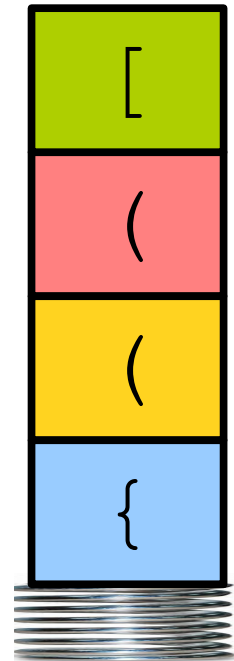
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





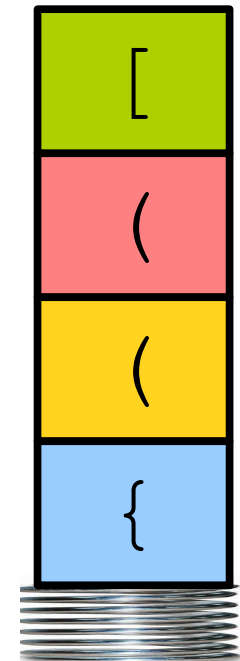
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



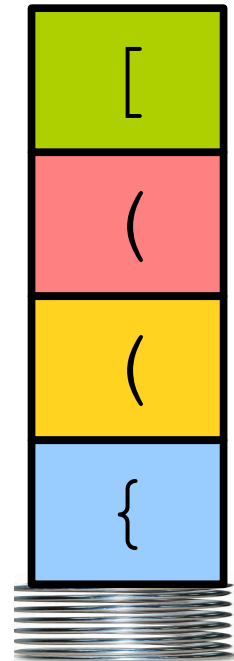
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



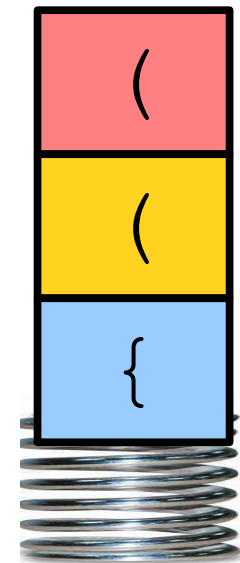
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



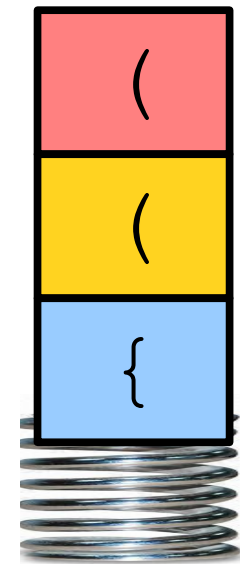
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



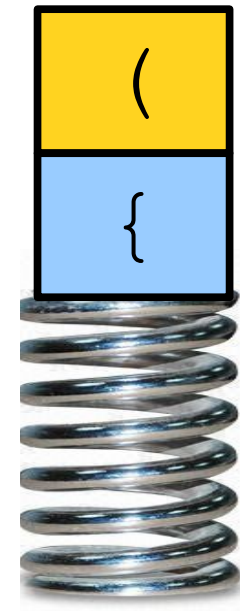
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



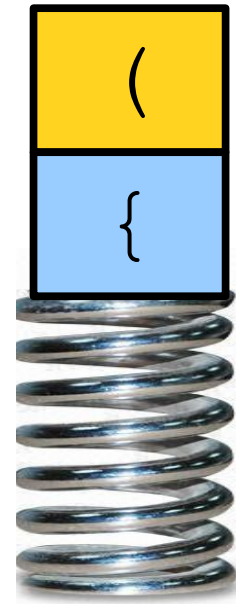
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



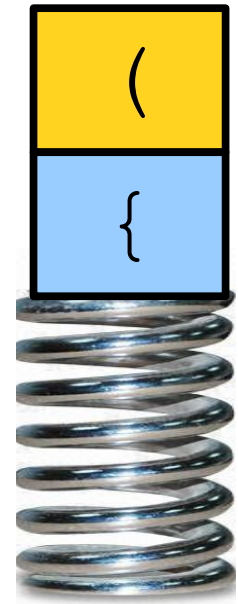
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

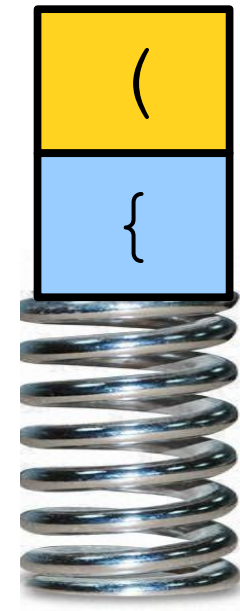
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





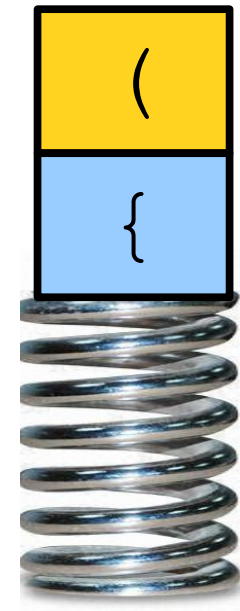
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



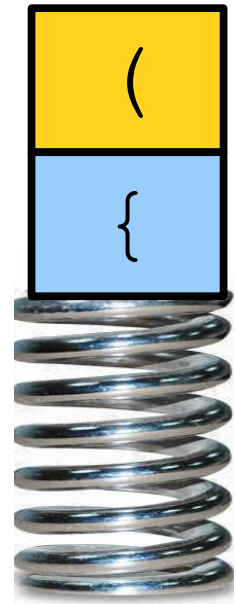
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



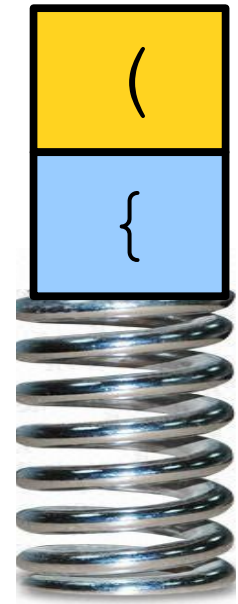
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



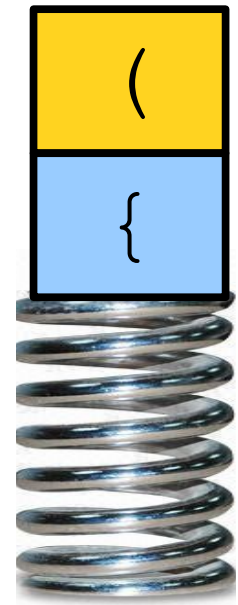
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

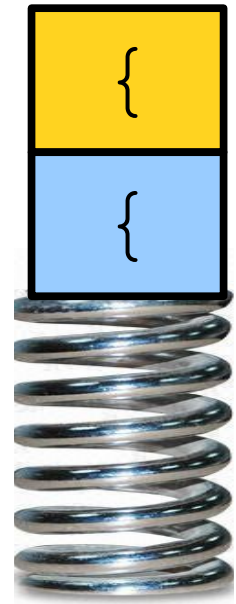
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





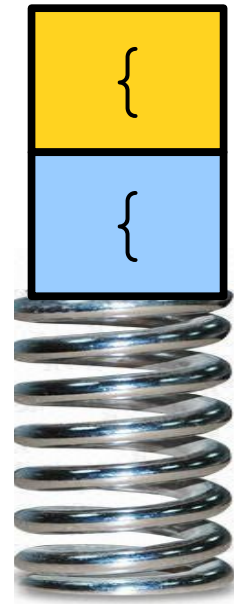
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



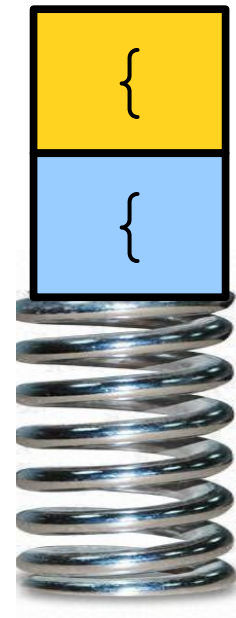
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



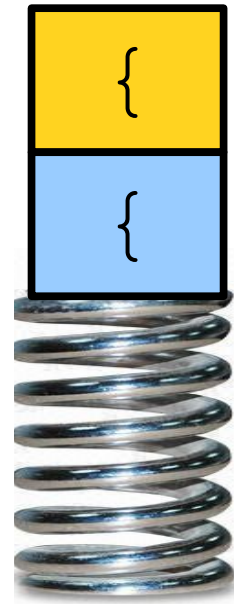
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



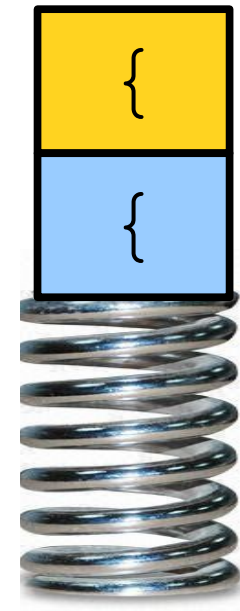
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



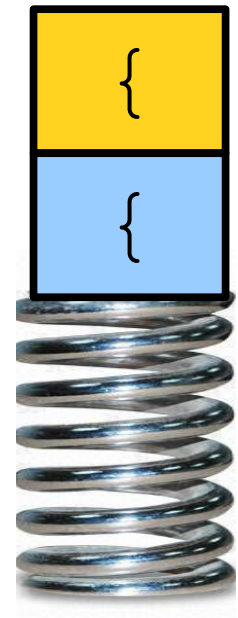
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



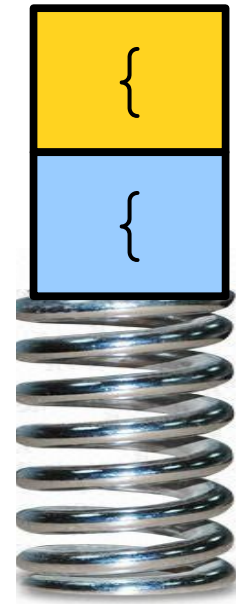
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



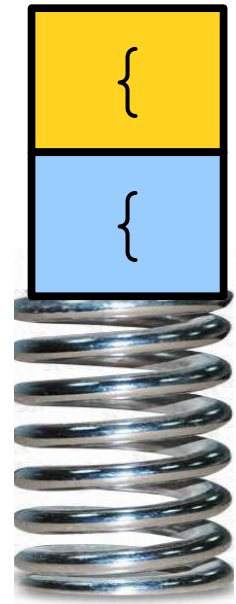
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

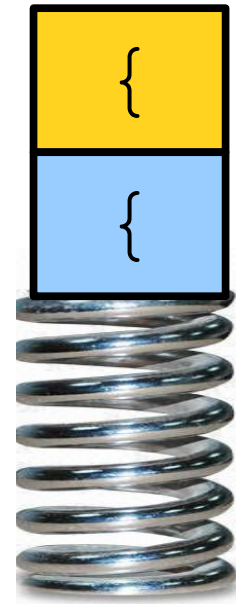
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





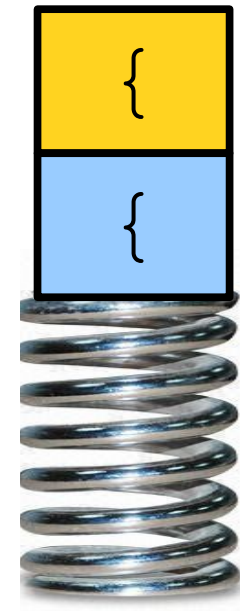
# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
                                                ^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }  
                                                ^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } } ^
```



# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```





# Balancing Parentheses

```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
```

Balancing Parentheses  
(On Board)



# A Few Functions to Make Life Easier...

```
bool isLeftParen(char ch) {
    return ch == '(' || ch == '{' || ch '[';
}

bool isRightParen(char ch) {
    return ch == ')' || ch == '}' || ch ']';
}

bool isMatchingParen(char left, char right) {
    return (left == '(' && right == ')') ||
           (left == '[' && right == ']') ||
           (left == '{' && right == '}');
}
```

**Combining** TokenScanner **and** Stack:  
Evaluating Expressions

# Evaluating Expressions

- We want to be able to evaluate simple arithmetic expressions composed of integers and the four basic arithmetic operators “+,-,\*,/”
  - $5 * 20 - 8 + 5$
- Proposed algorithm: just evaluate the expression from left to right.
  - $5 * 8 + 7 = 40 + 7 = 47$
  - $1 + 2 + 4 = 3 + 4 = 7$
- It works...or does it?
  - $7 + 5 * 8 = 12 * 8 = 96???$

# Evaluating Expressions

- Evaluating expressions is much trickier than it might seem due to issues of precedence.
  - $1 + 3 * 5 - 7 = 9$
- We can't just evaluate operators from left to right
- How do we evaluate an expression?

# The Challenge

1	3	7		+		4	2		×		2	7	1
---	---	---	--	---	--	---	---	--	---	--	---	---	---

# Evaluating Expressions

- Two separate concerns in evaluating expressions:
  - **Scanning** the string and breaking it apart into its constituent components (*tokens*).
  - **Parsing** the tokens to determine what expression is encoded.
- We can scan the string with the `TokenScanner`. How might we handle parsing?

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



**Operands**

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



**Operands**



**Operators**

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



**Operands**



**Operators**

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



Operands

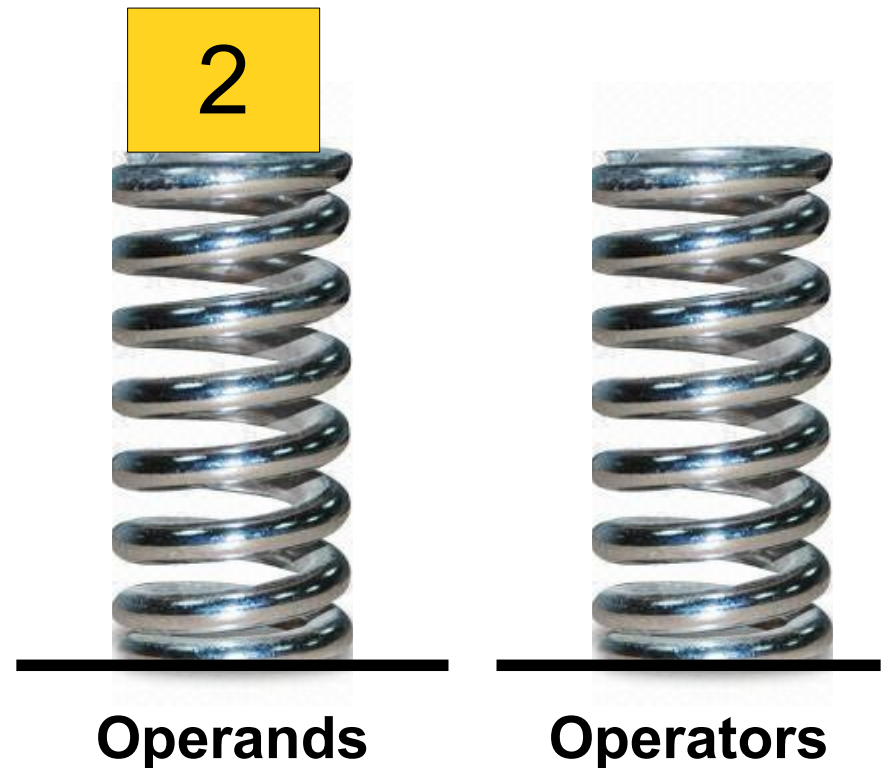


Operators

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

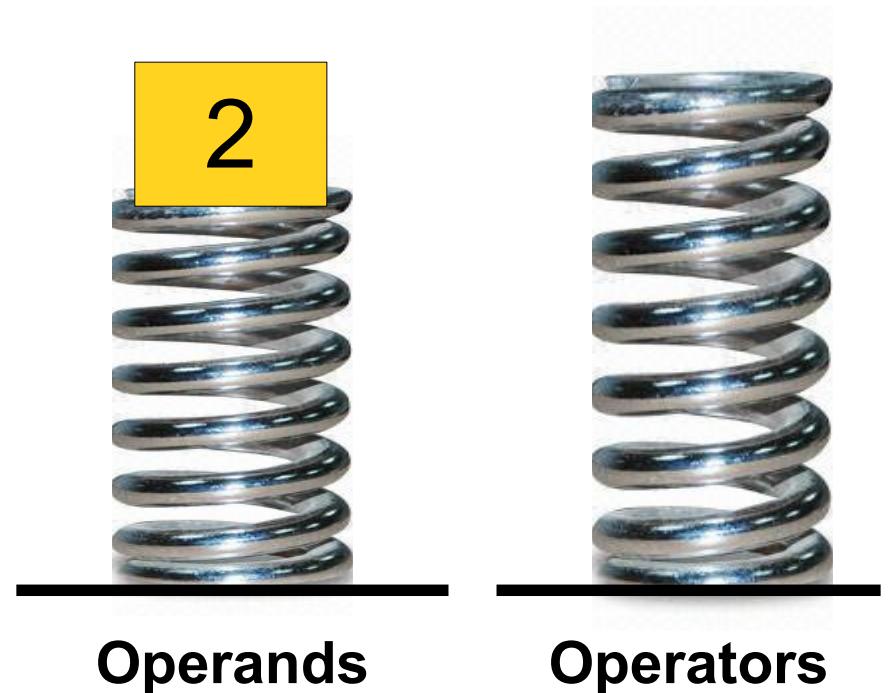
+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

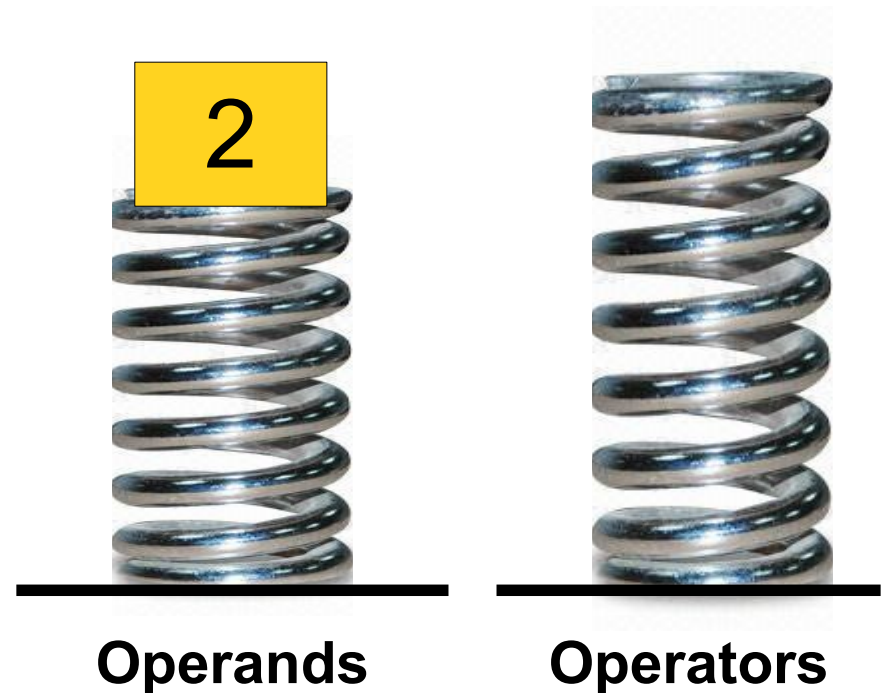
+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

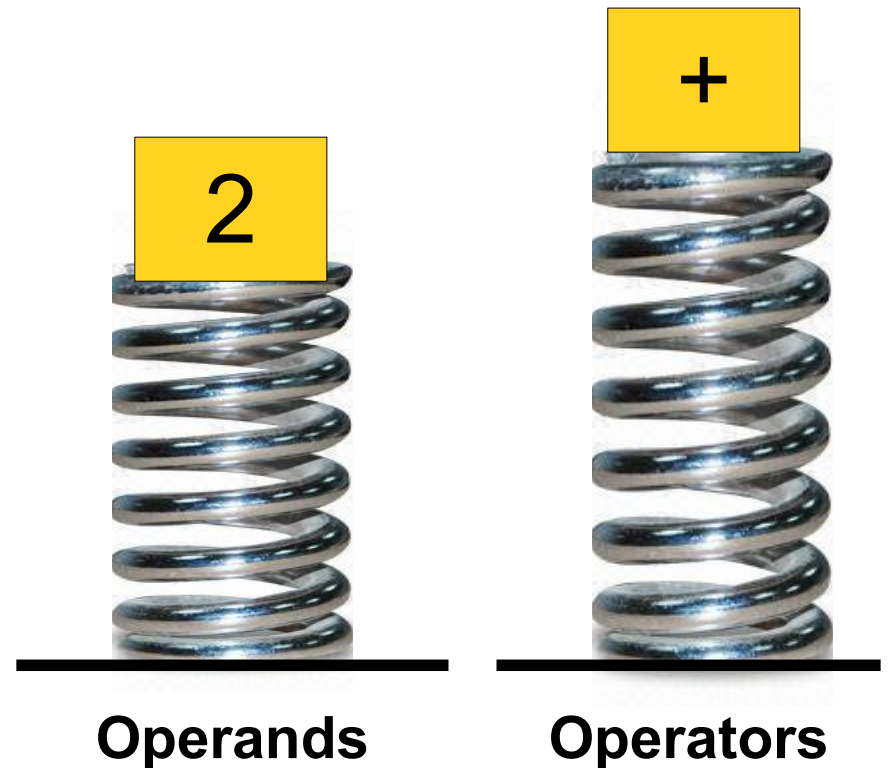
+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

3	*	5	-	6	/	2
---	---	---	---	---	---	---

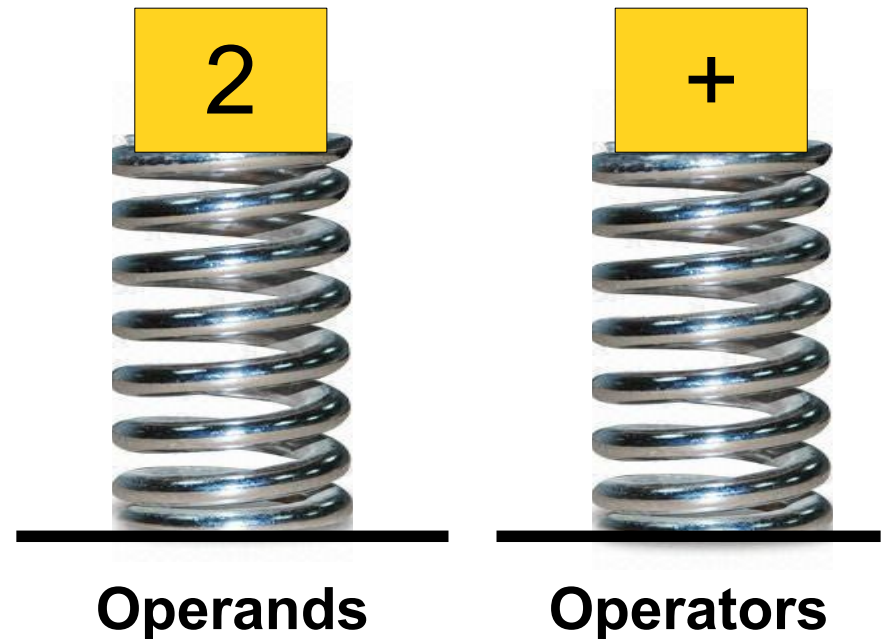




# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

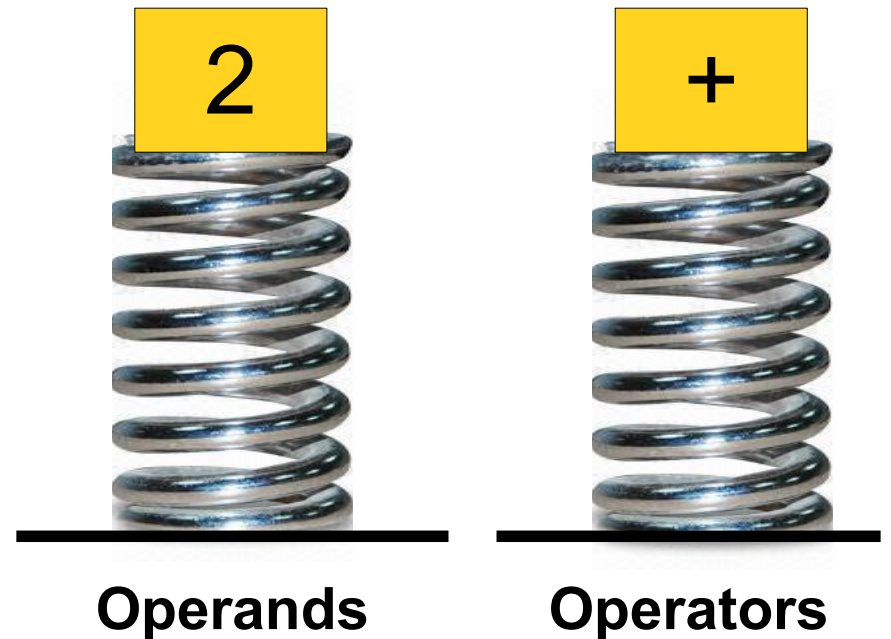
3	*	5	-	6	/	2
---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

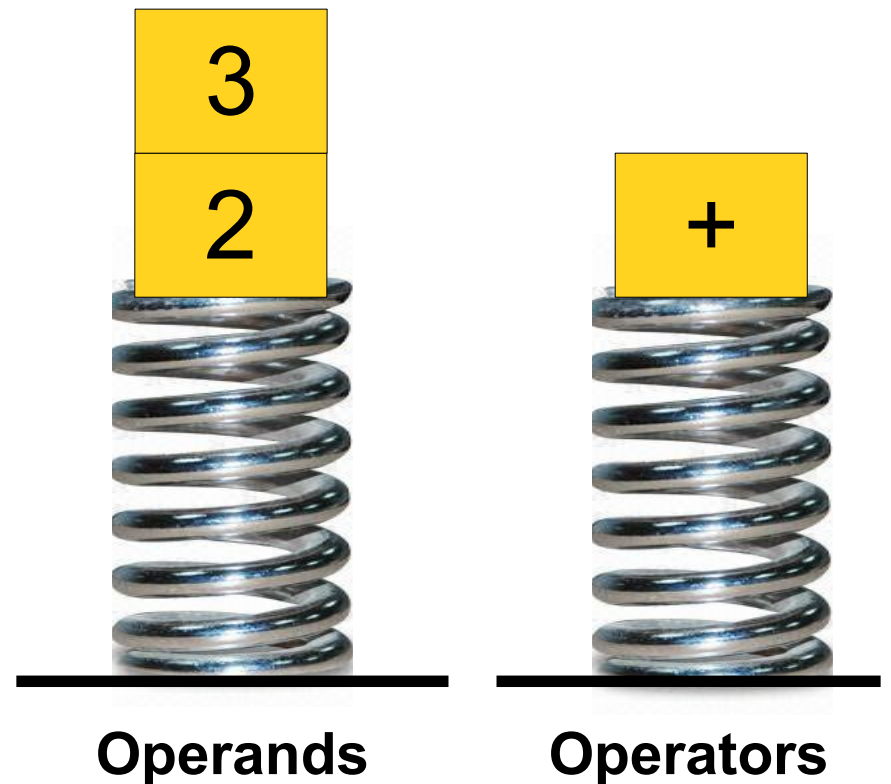
3	*	5	-	6	/	2
---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

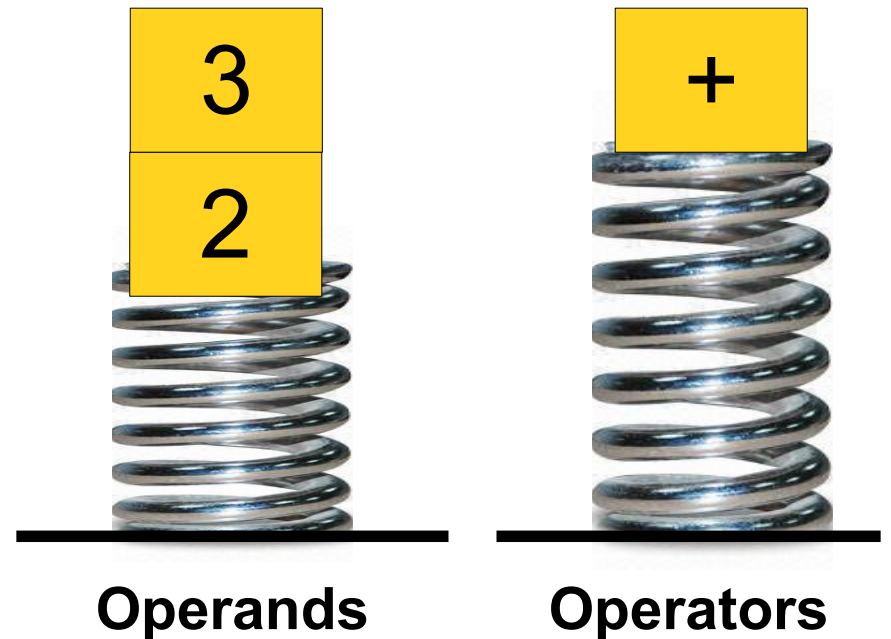
*	5	-	6	/	2
---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

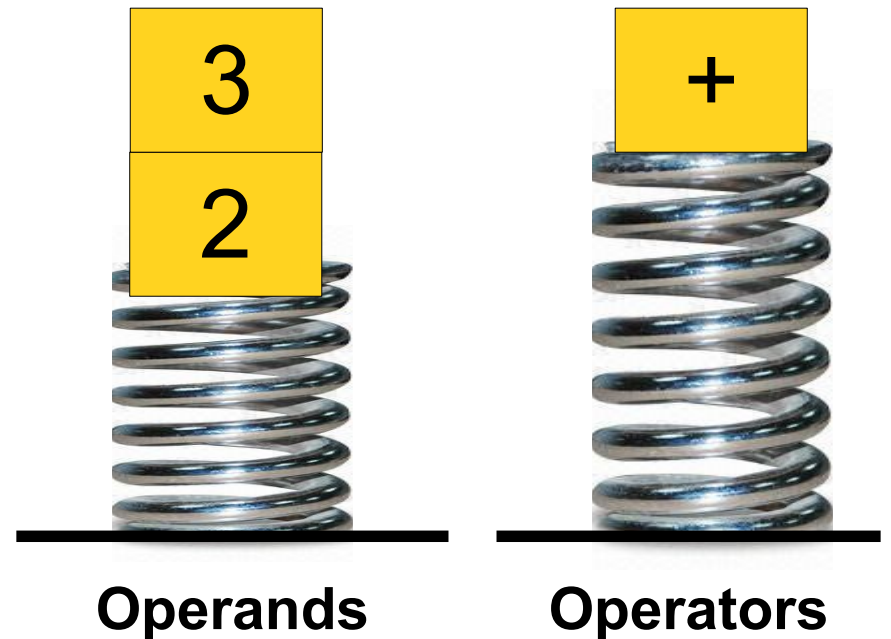
*	5	-	6	/	2
---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

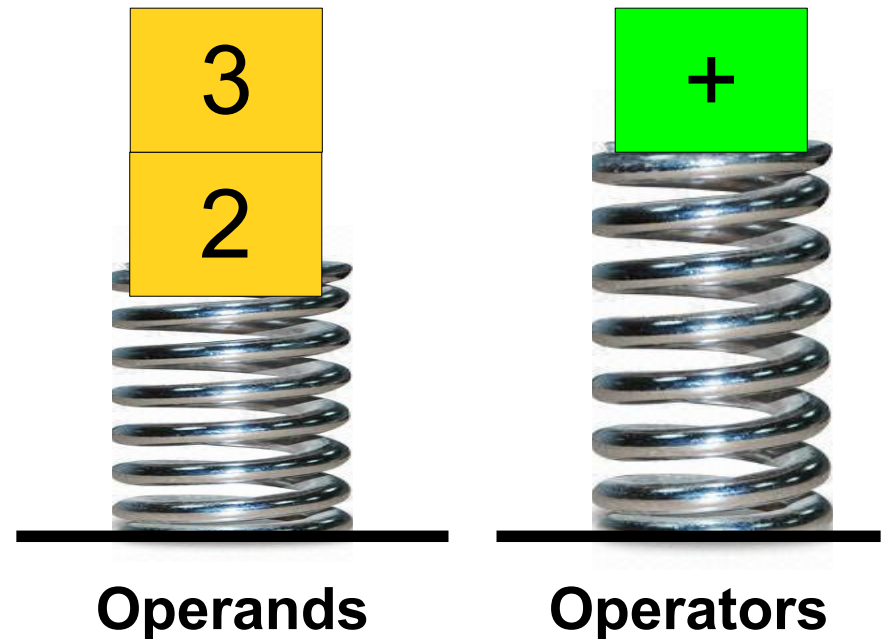
*	5	-	6	/	2
---	---	---	---	---	---



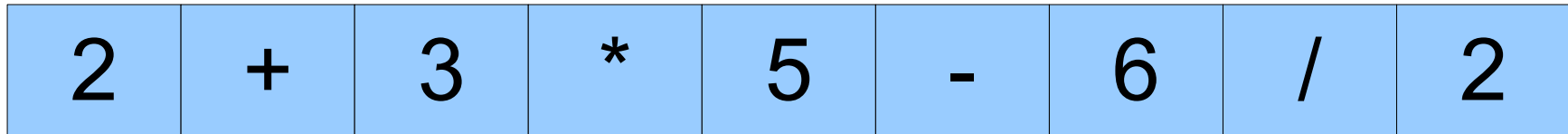
# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

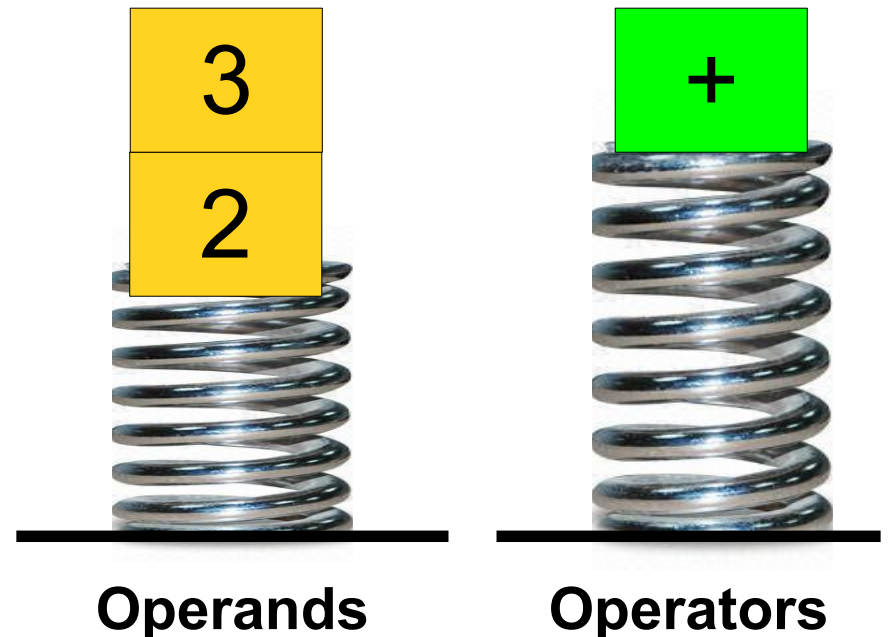
*	5	-	6	/	2
---	---	---	---	---	---



# The Shunting-Yard Algorithm



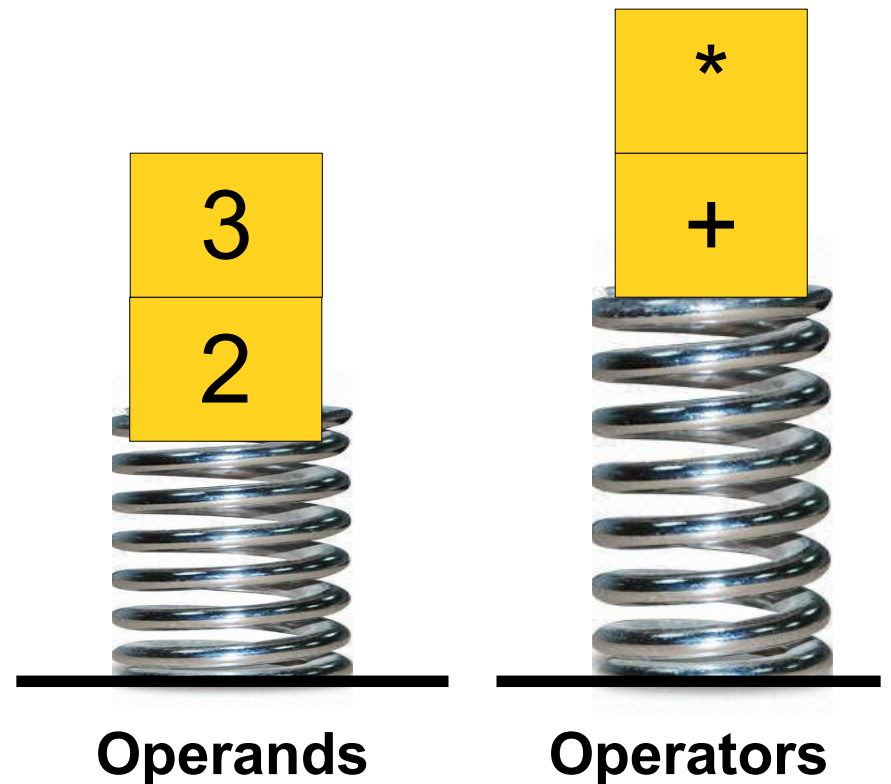
Multiplication has higher precedence than addition, so we will postpone the addition until after we've done the multiplication.



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

5	-	6	/	2
---	---	---	---	---

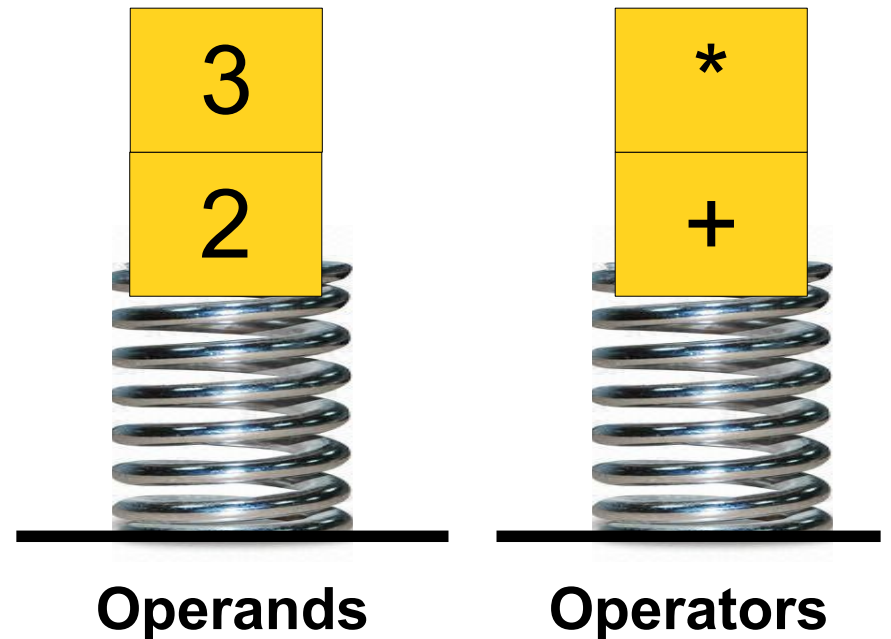




# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

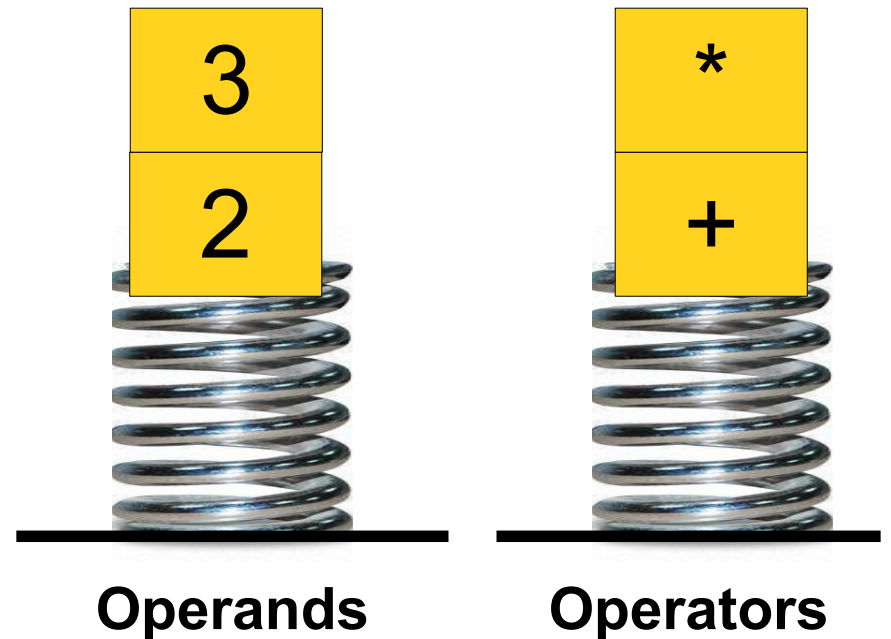
5	-	6	/	2
---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

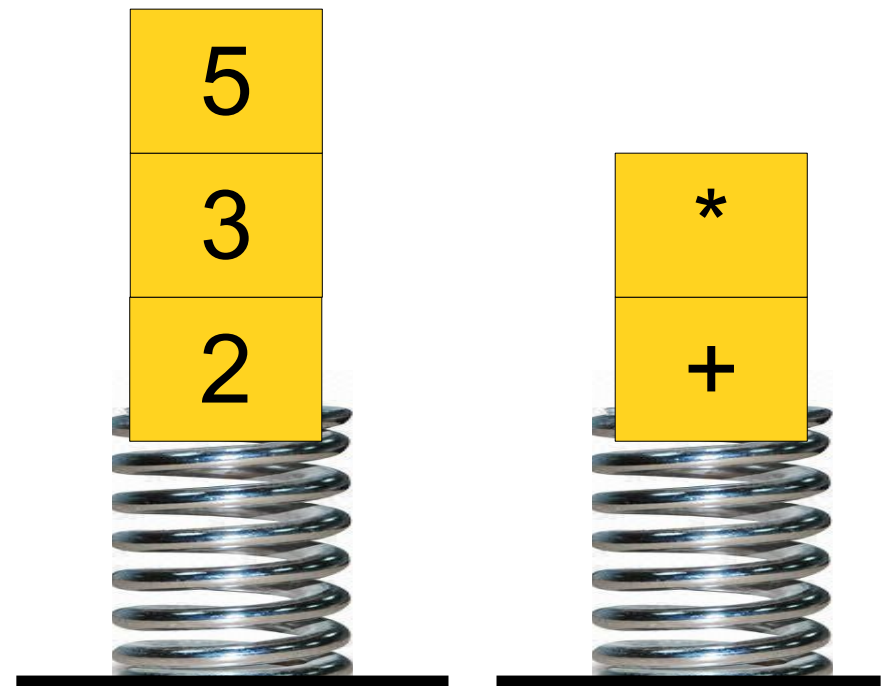
5	-	6	/	2
---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

-	6	/	2
---	---	---	---



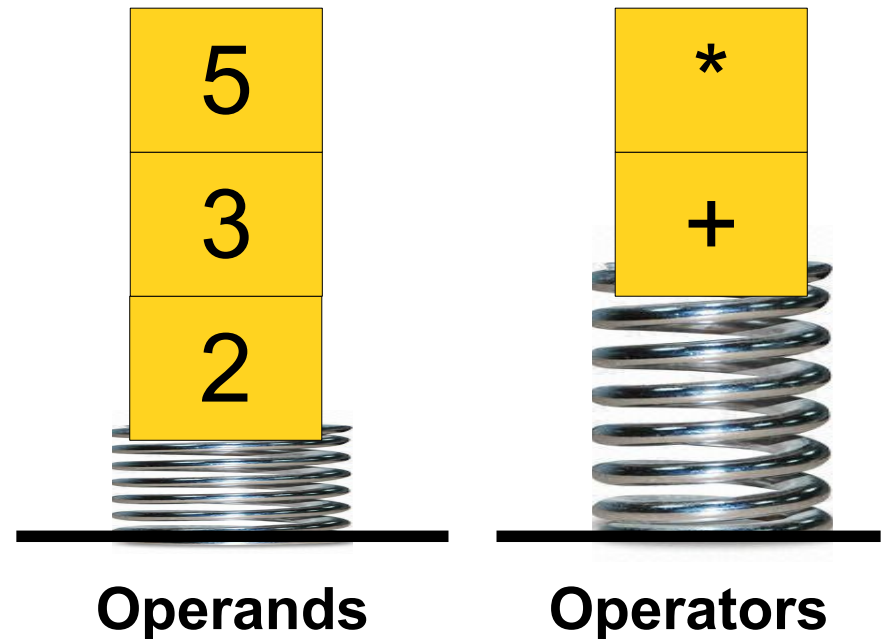
Operands

Operators

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

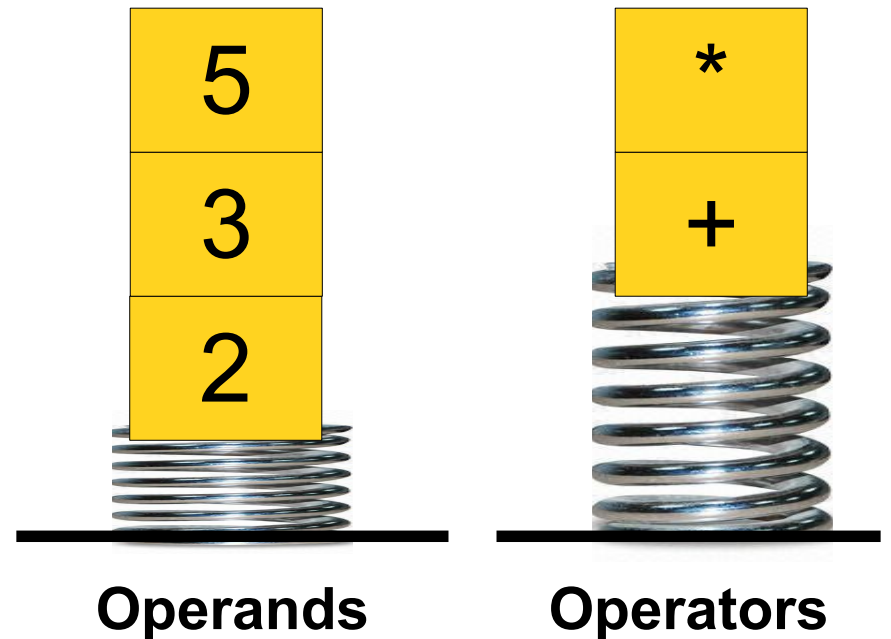
-	6	/	2
---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

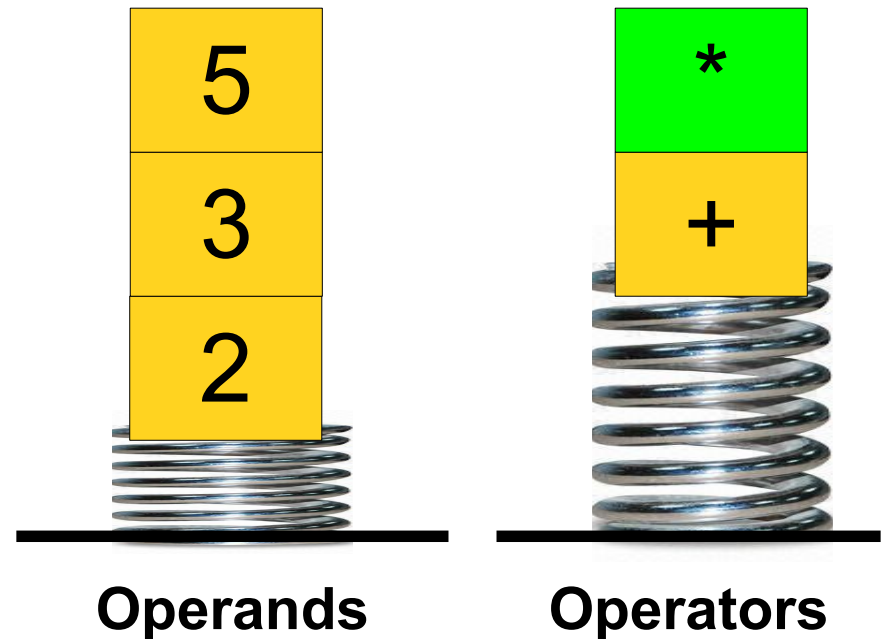
-	6	/	2
---	---	---	---



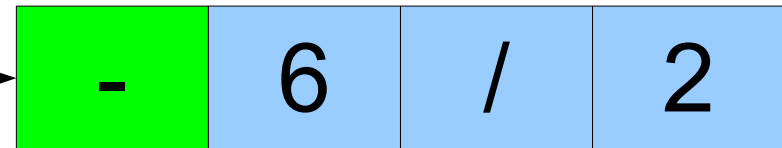
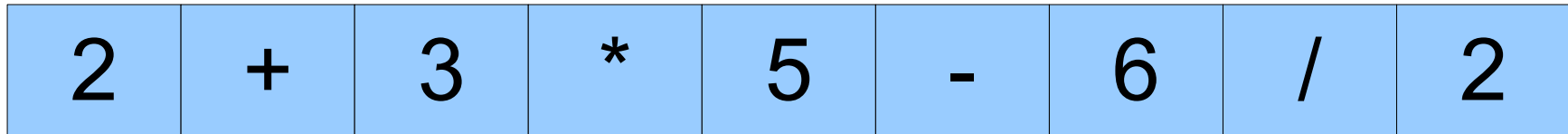
# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

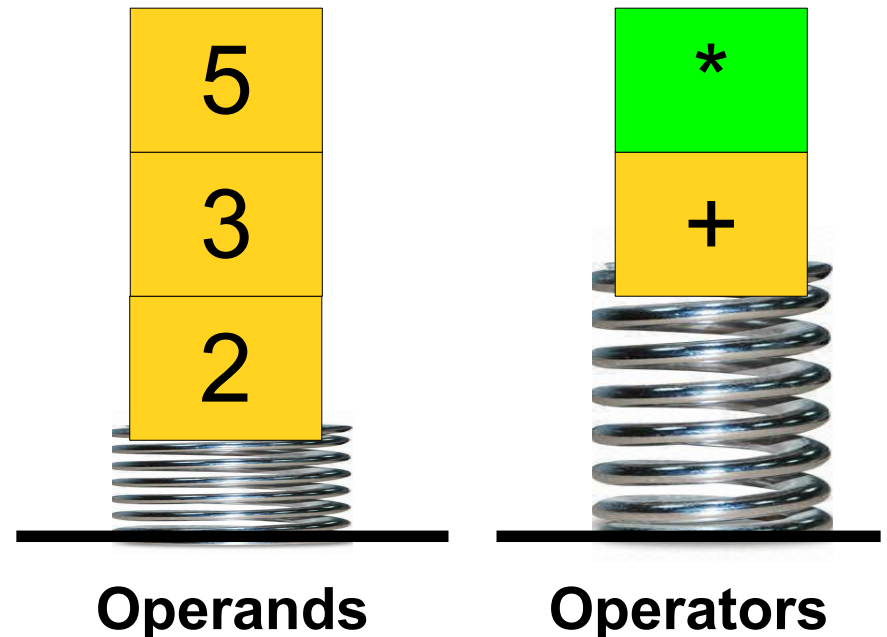
-	6	/	2
---	---	---	---



# The Shunting-Yard Algorithm



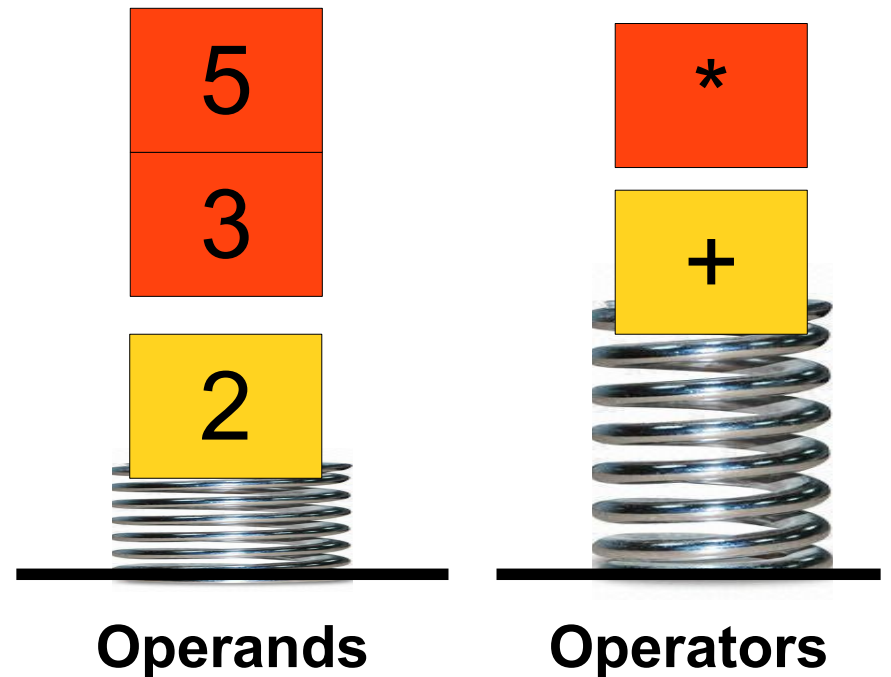
Subtraction has lower precedence than multiplication, so we need to evaluate the multiply before the subtract.



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

-	6	/	2
---	---	---	---



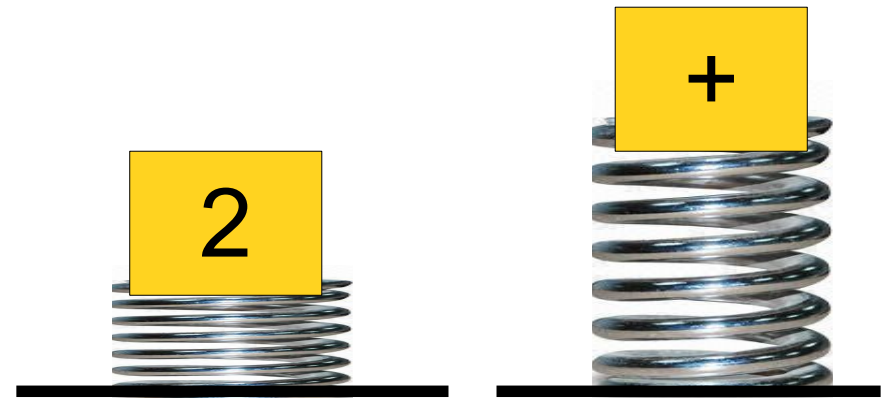


# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

-	6	/	2
---	---	---	---

3	*	5
---	---	---



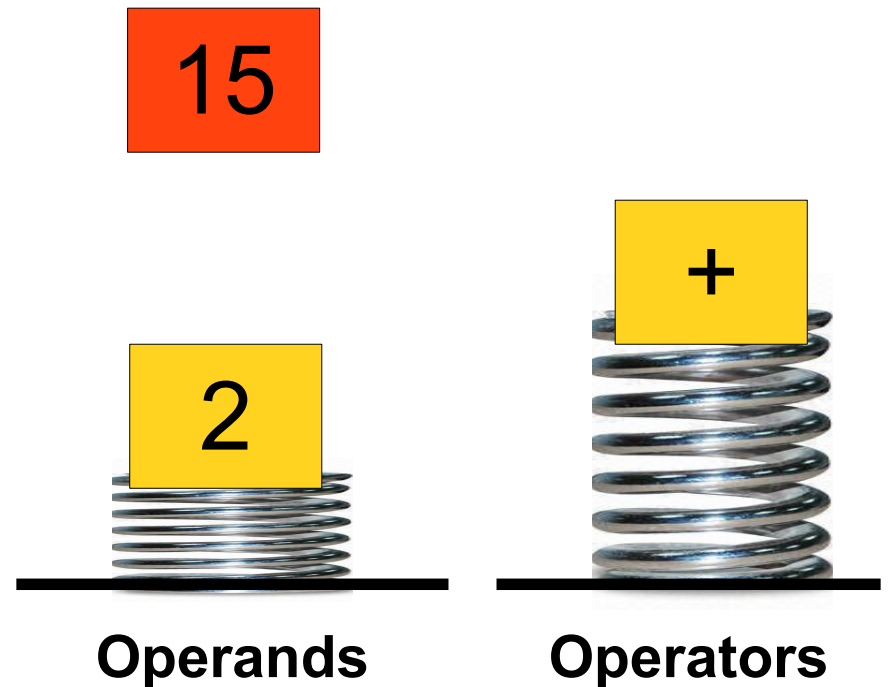
Operands

Operators

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

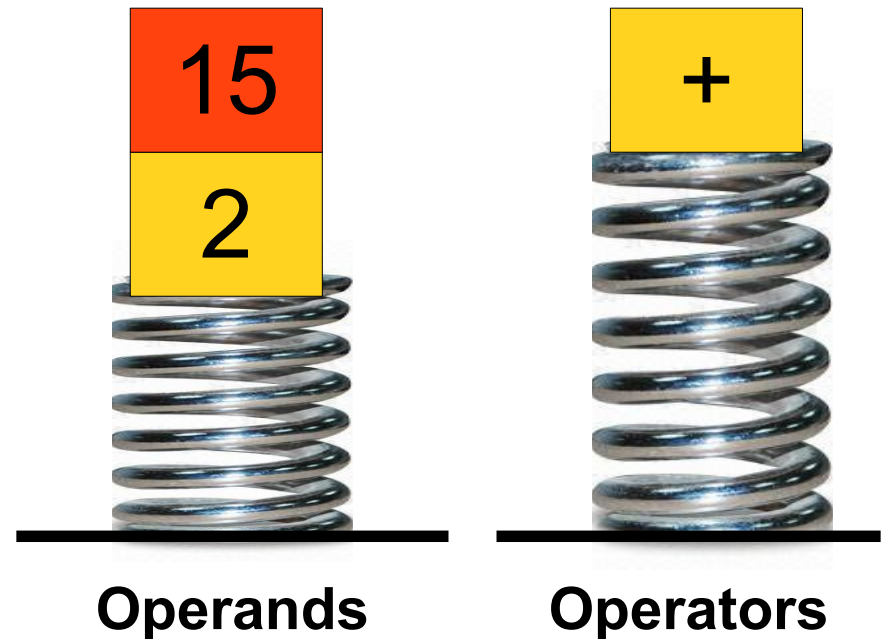
-	6	/	2
---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

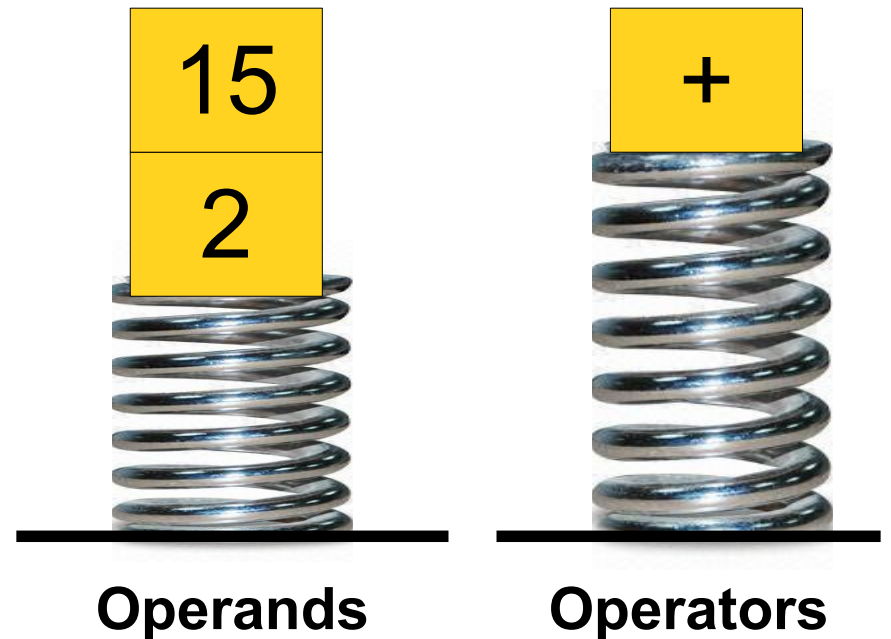
-	6	/	2
---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

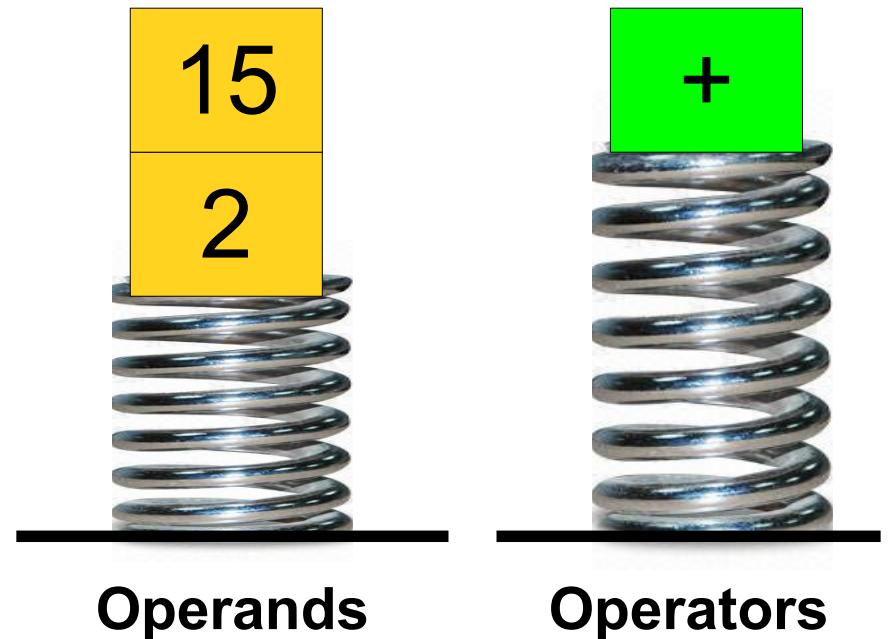
-	6	/	2
---	---	---	---



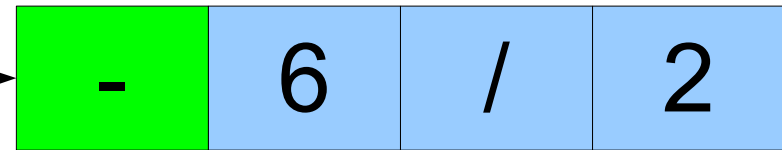
# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

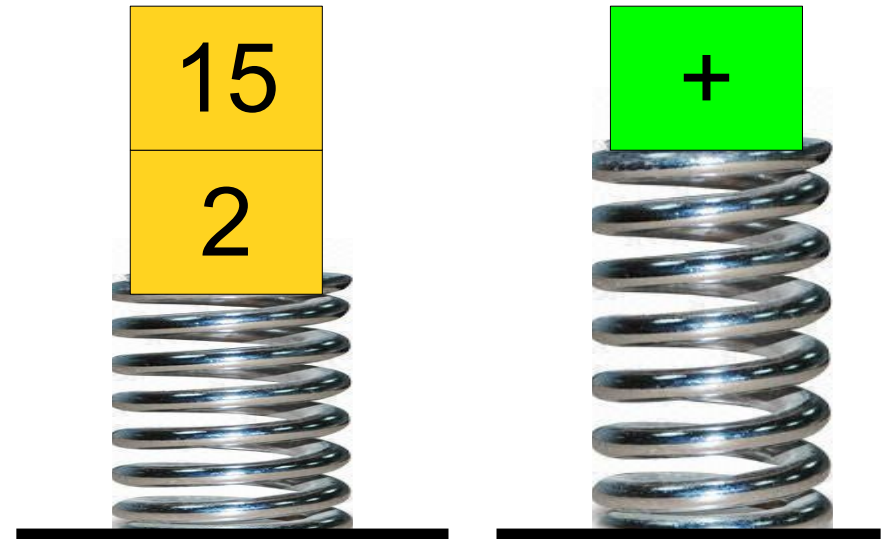
-	6	/	2
---	---	---	---



# The Shunting-Yard Algorithm



Subtraction has equal precedence to addition so we evaluate the add before the subtract.



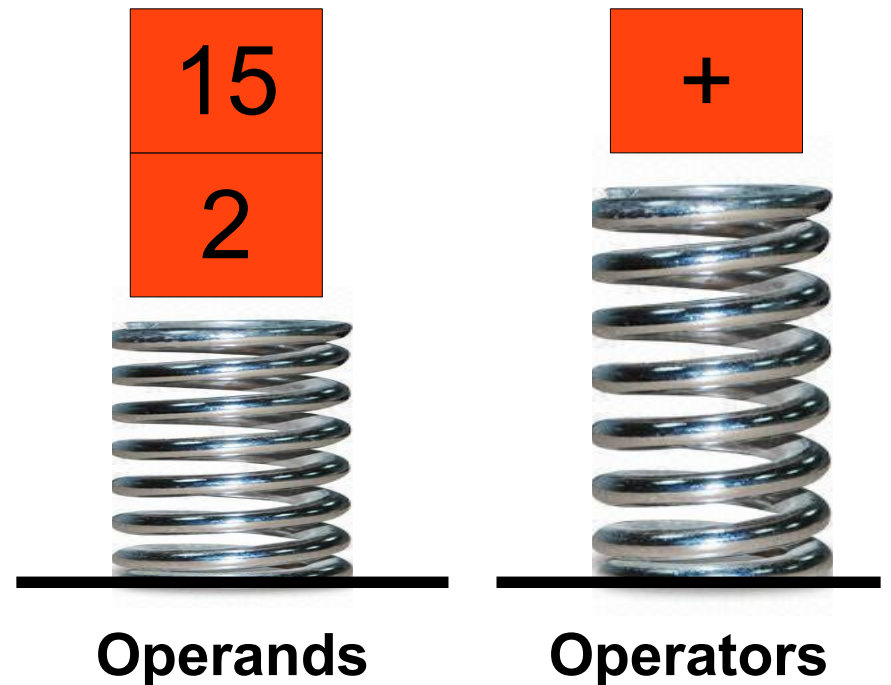
Operands

Operators

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

-	6	/	2
---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

-	6	/	2
---	---	---	---

2	+	15
---	---	----



Operands



Operators



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

-	6	/	2
---	---	---	---

17
----



Operands

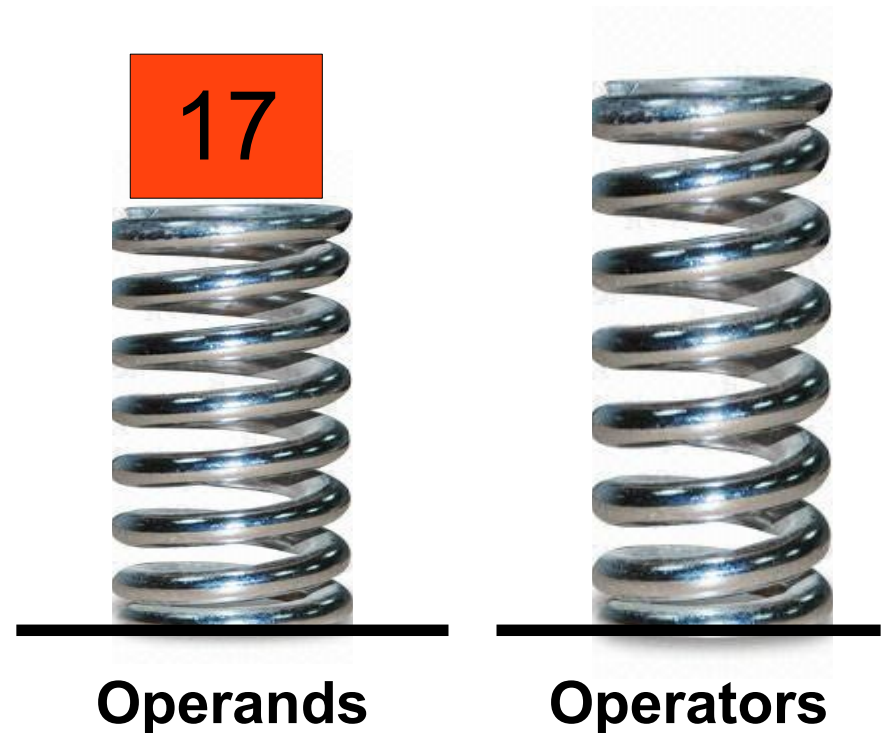


Operators

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

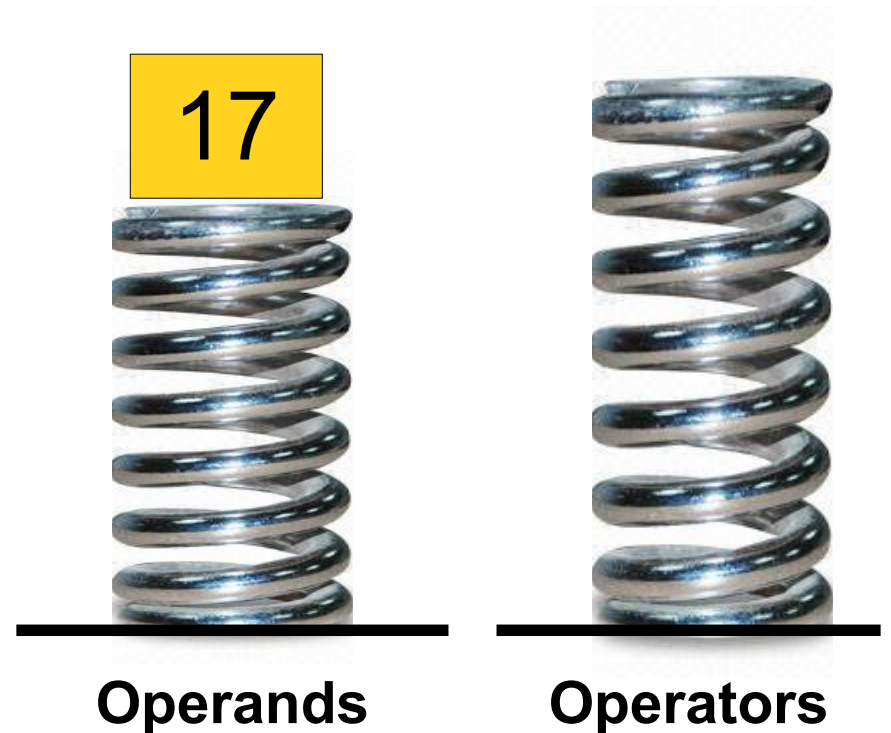
-	6	/	2
---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

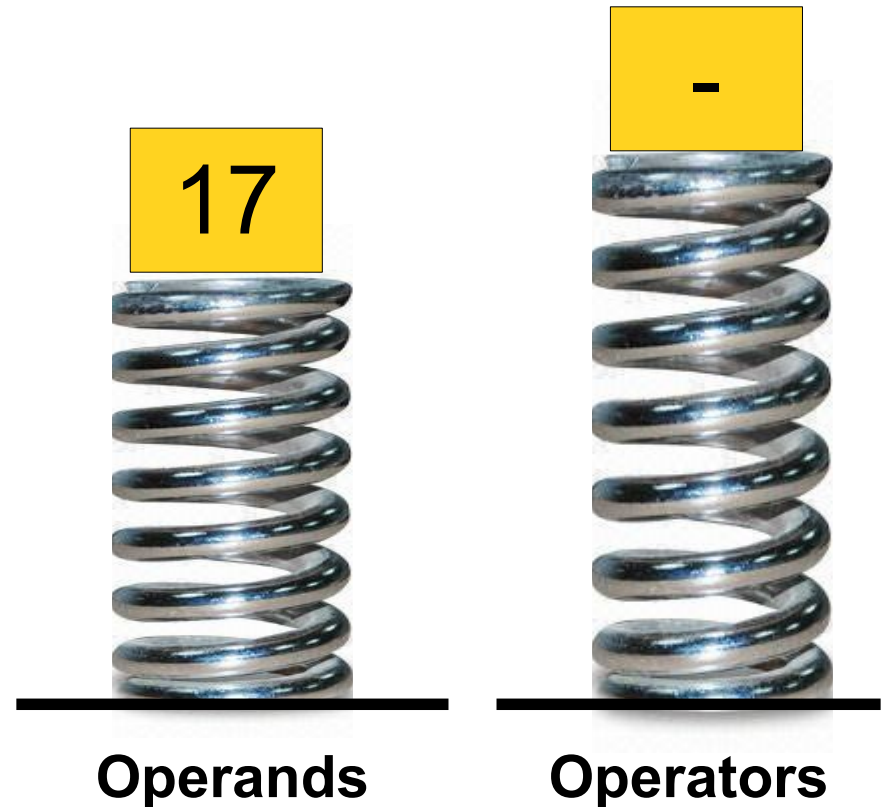
-	6	/	2
---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

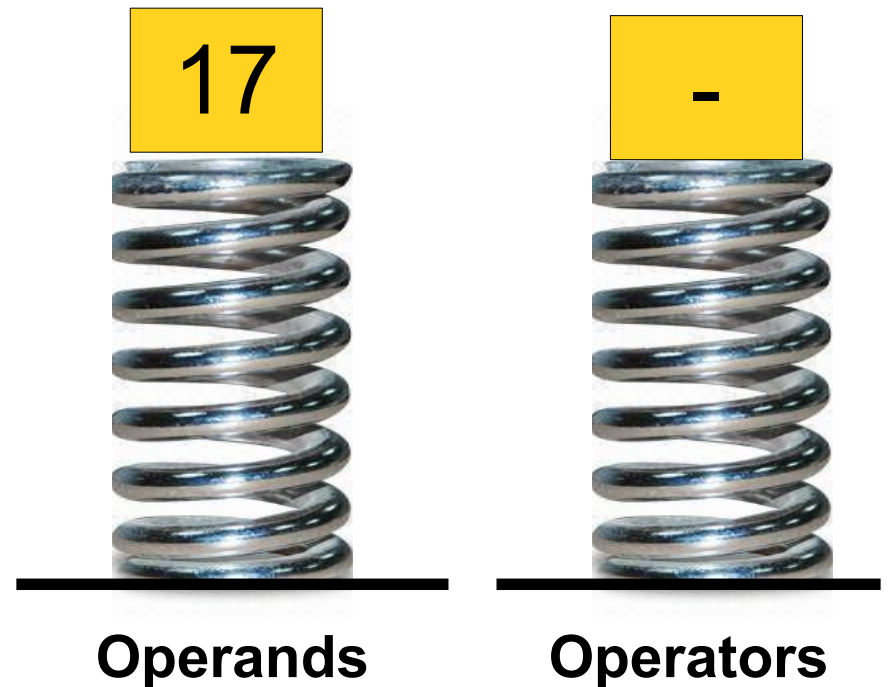
6	/	2
---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

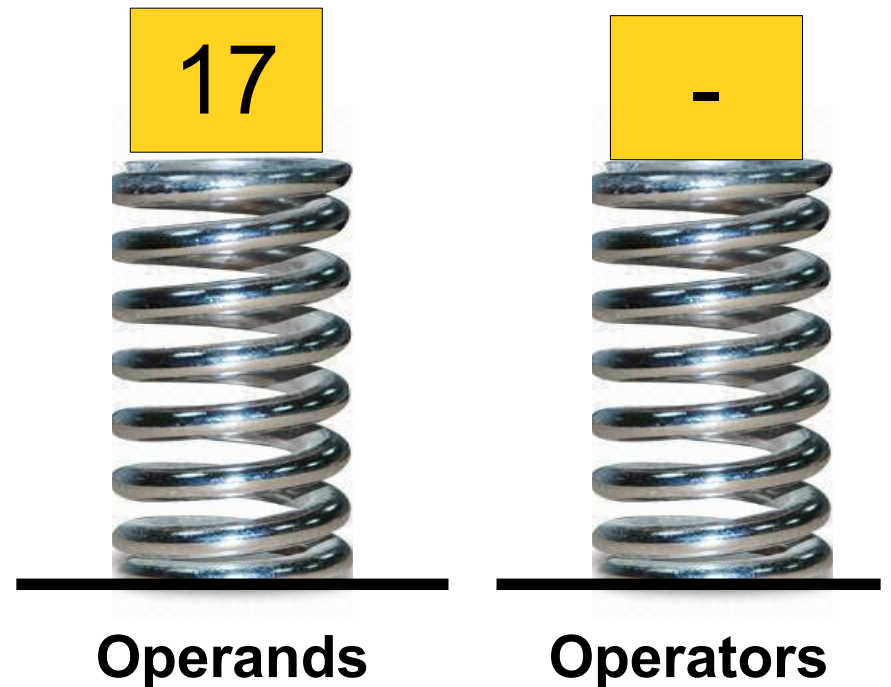
6	/	2
---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

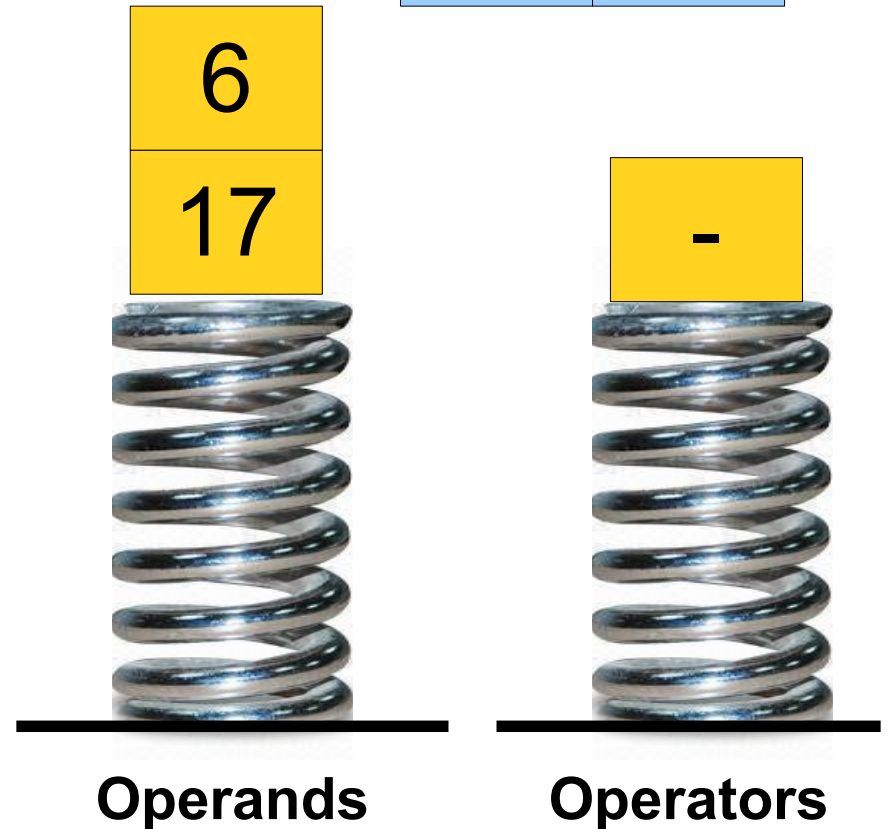
6	/	2
---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

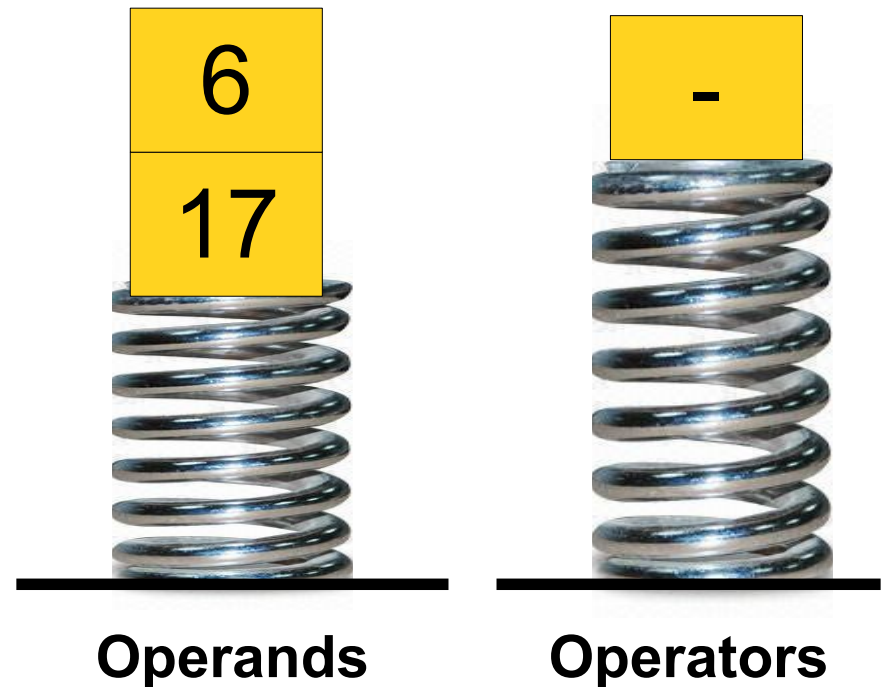
/	2
---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

/	2
---	---

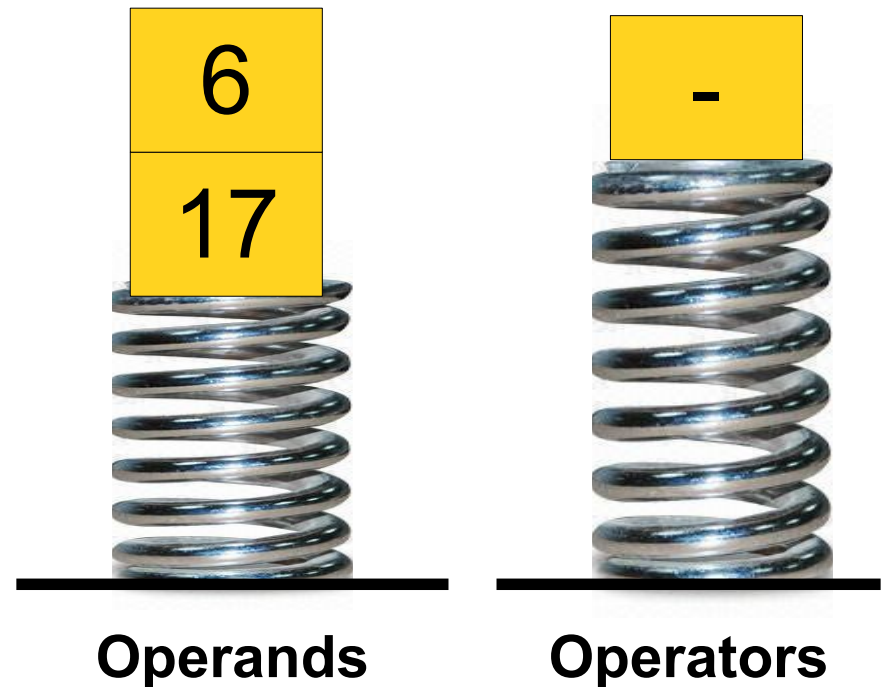




# The Shunting-Yard Algorithm

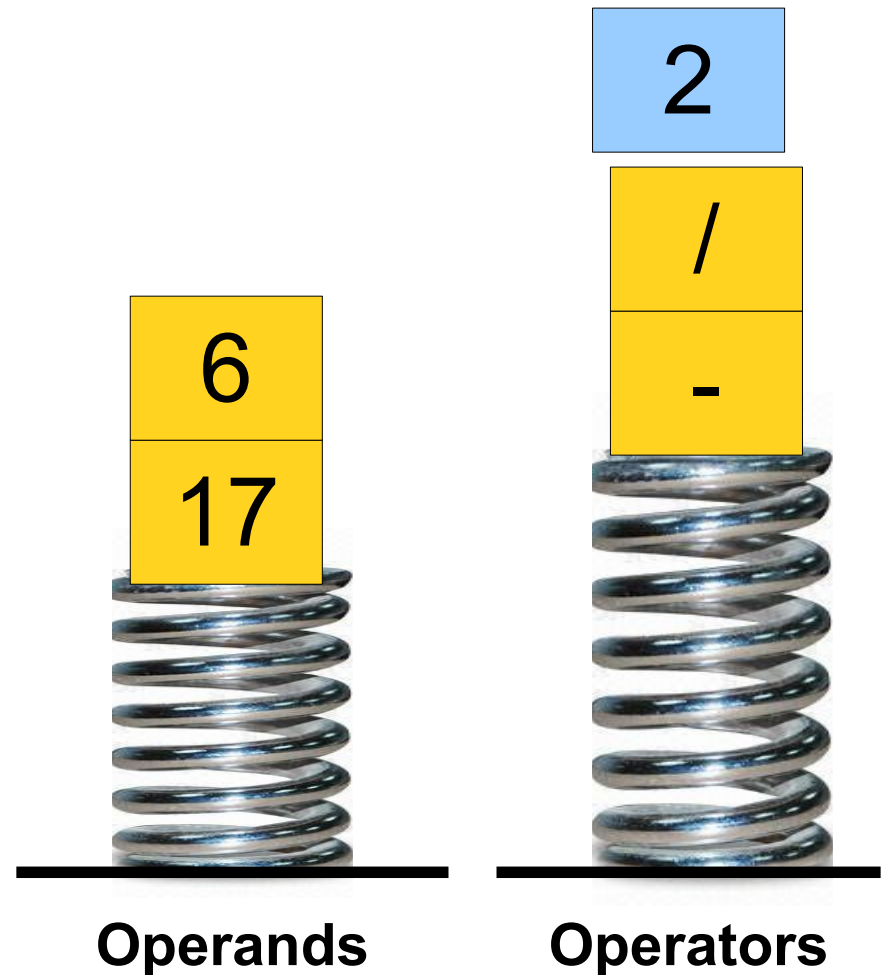
2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

/	2
---	---



# The Shunting-Yard Algorithm

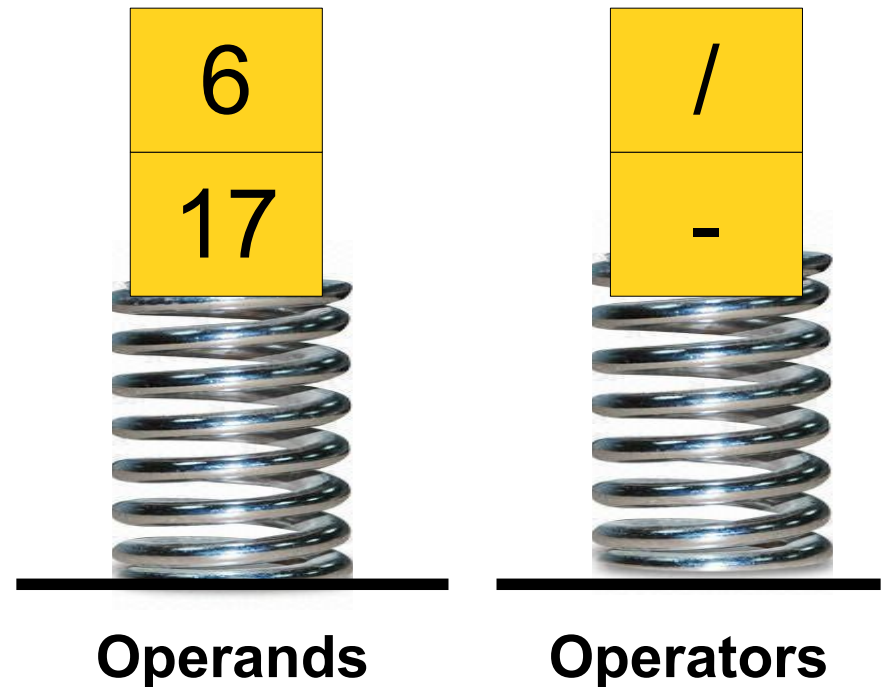
2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

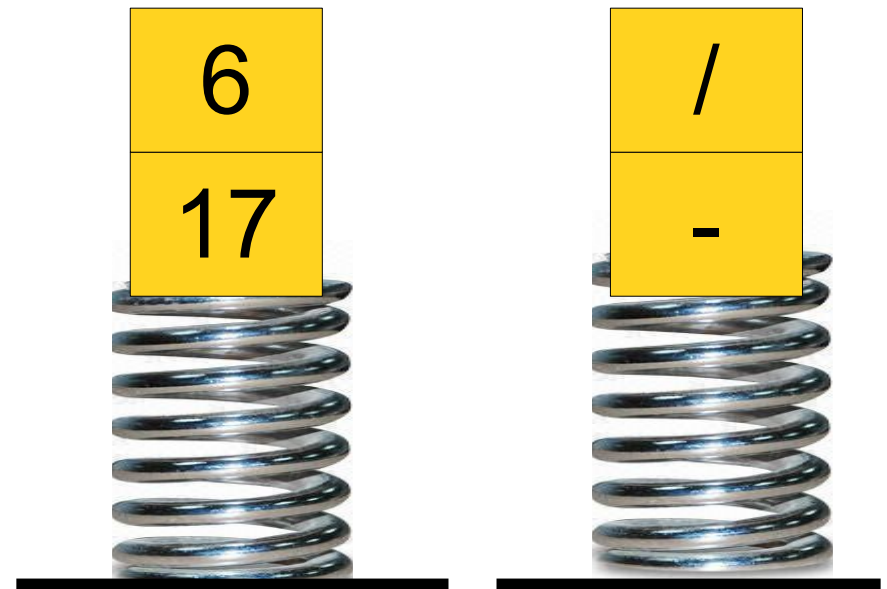
2
---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

2

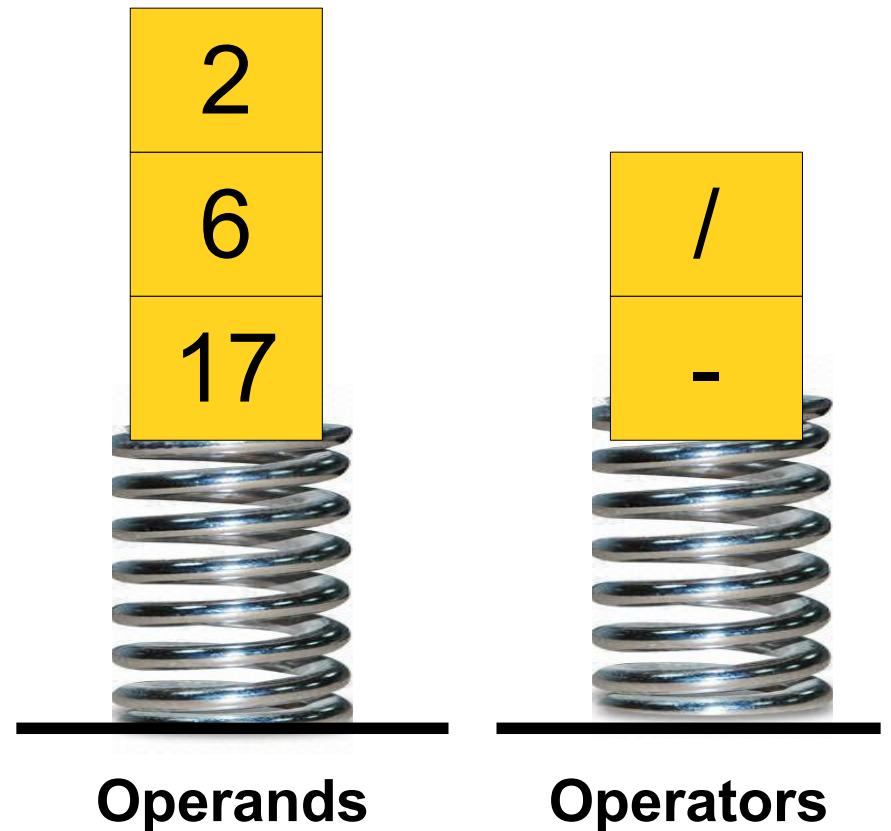


Operands

Operators

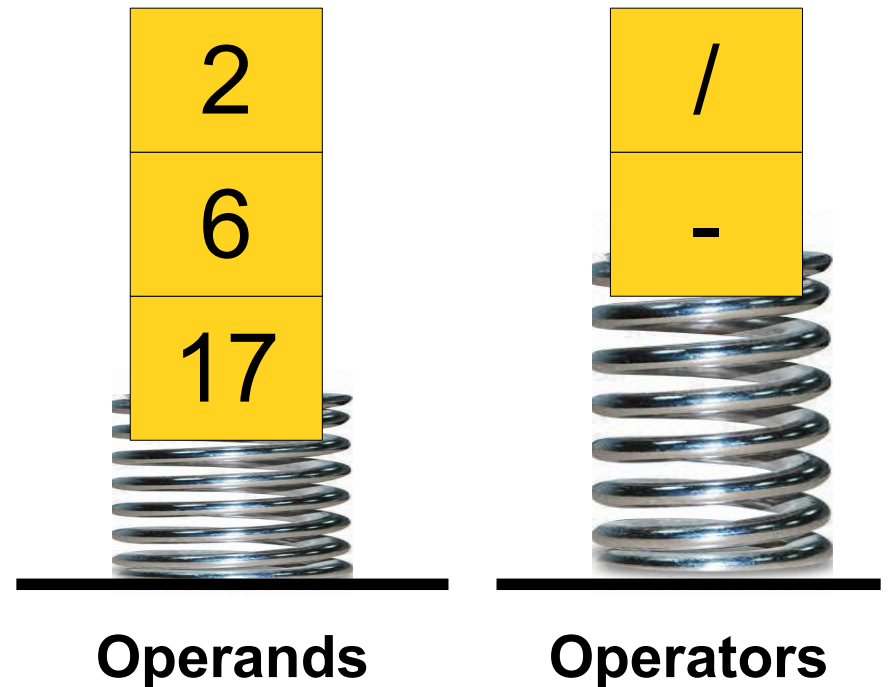
# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

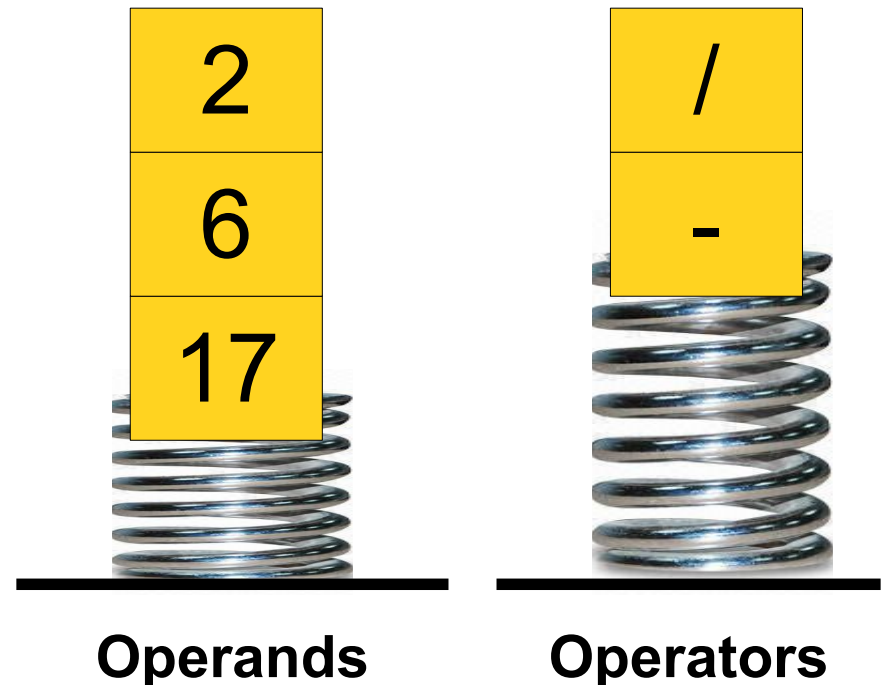
2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

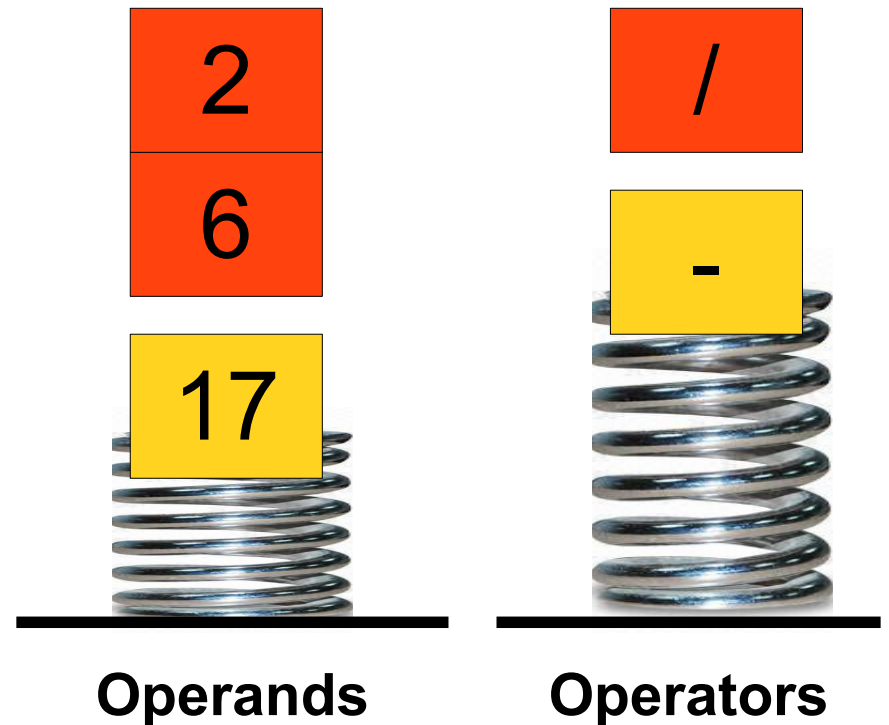
2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

Now that we've read all the tokens, we can finish evaluating all the expressions.



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

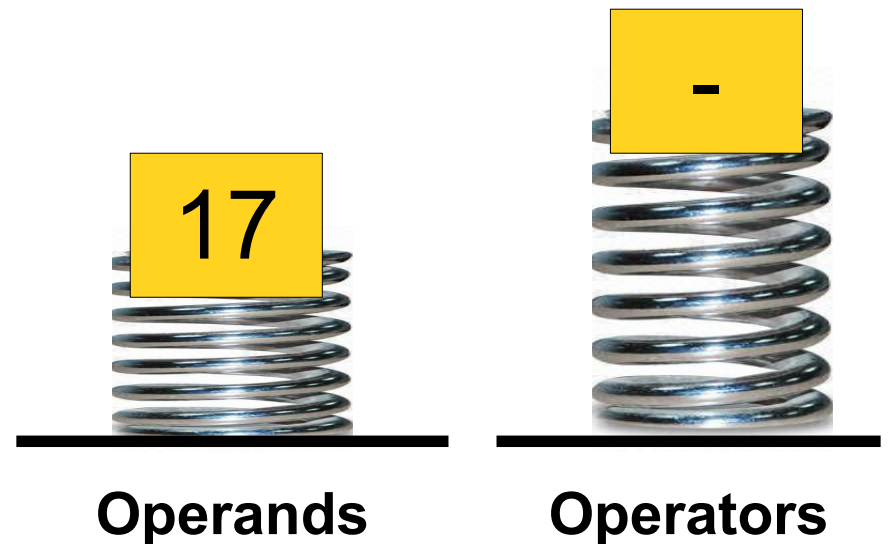




# The Shunting-Yard Algorithm

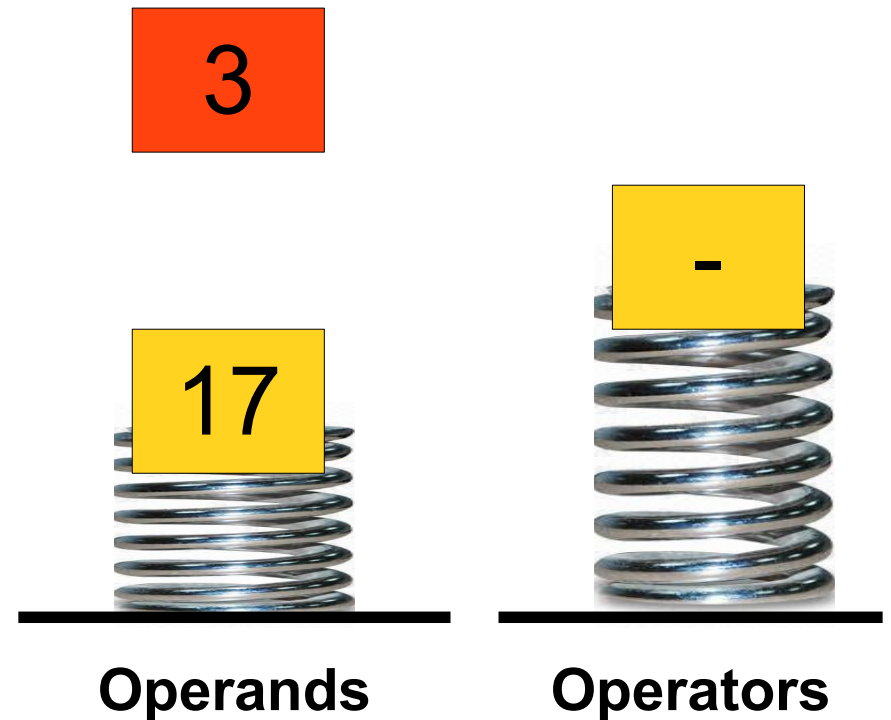
2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

6	/	2
---	---	---



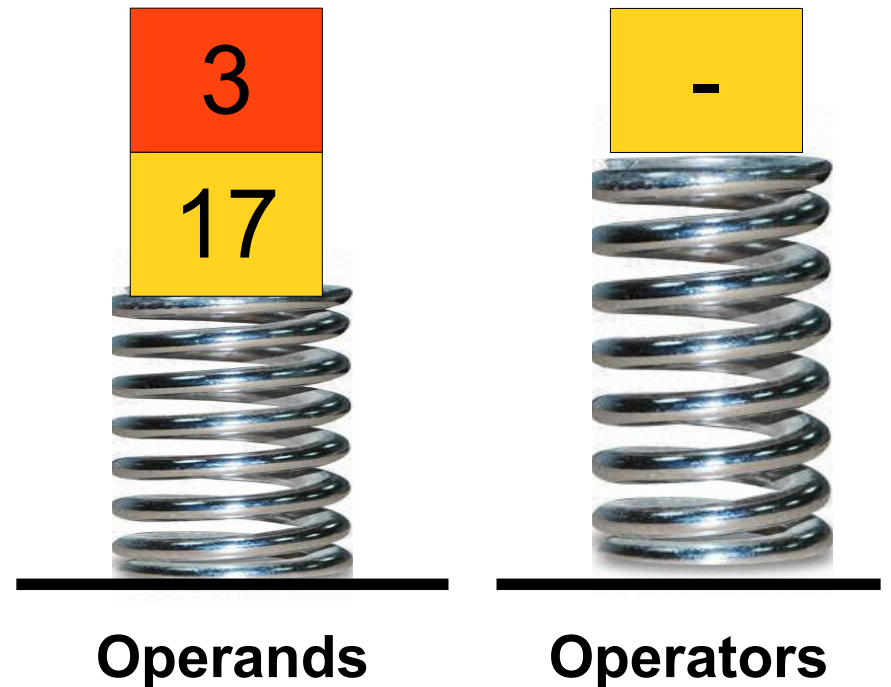
# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



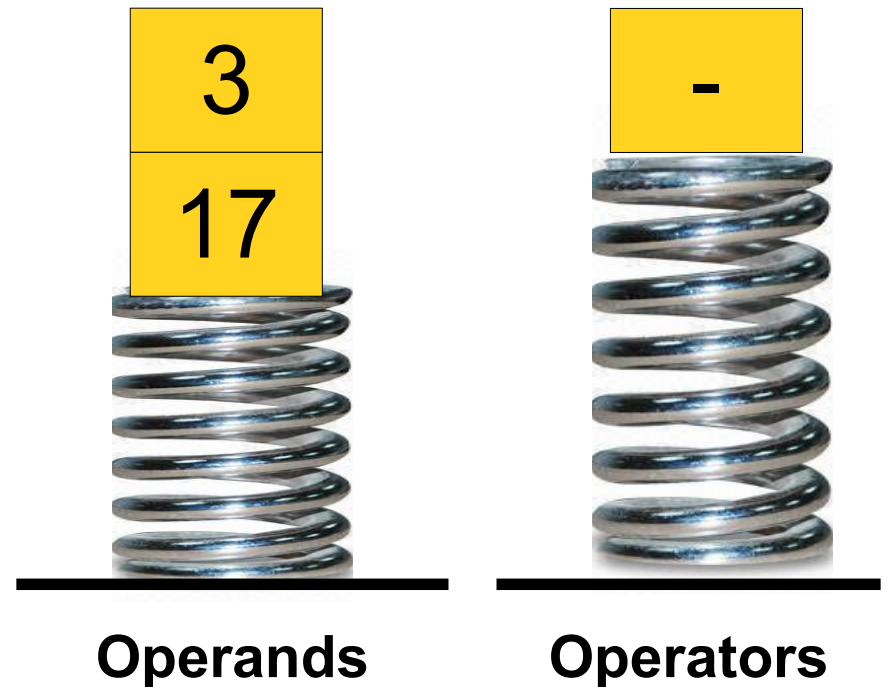
# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



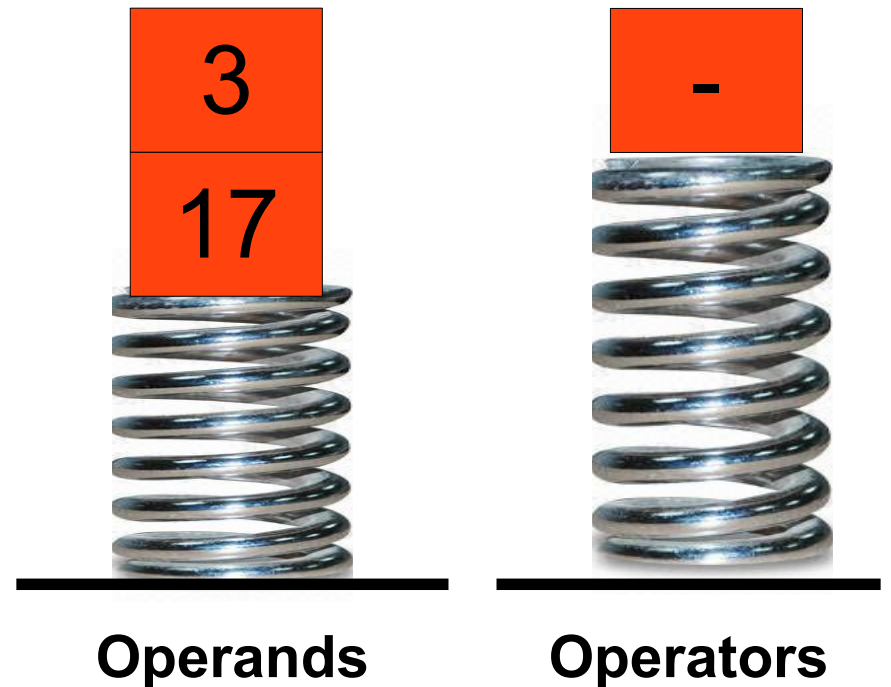
# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

17	-	3
----	---	---



**Operands**



**Operators**

# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---

14



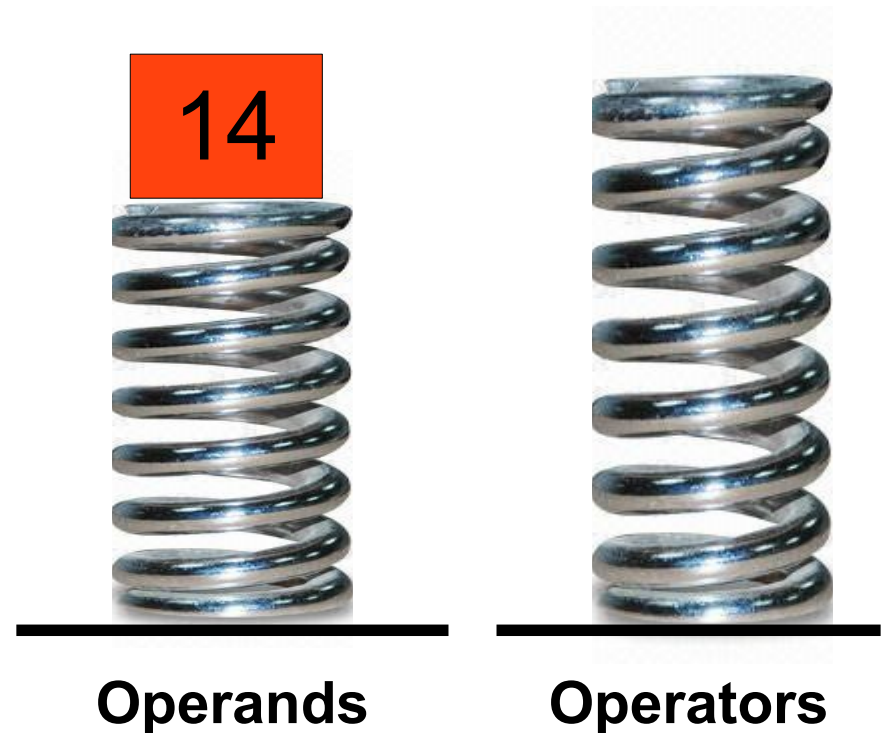
Operands



Operators

# The Shunting-Yard Algorithm

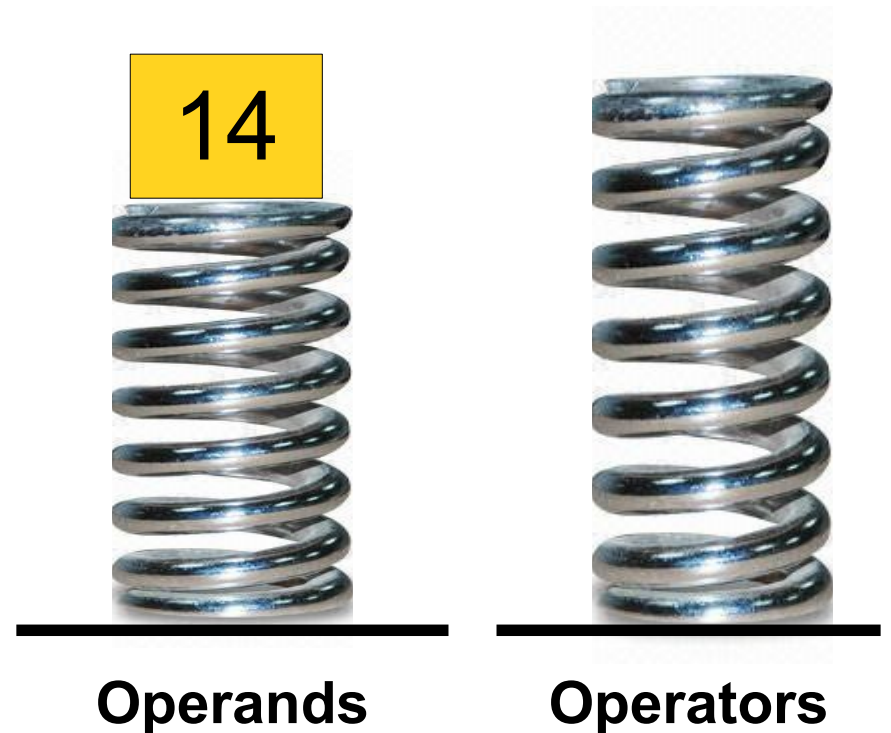
2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---





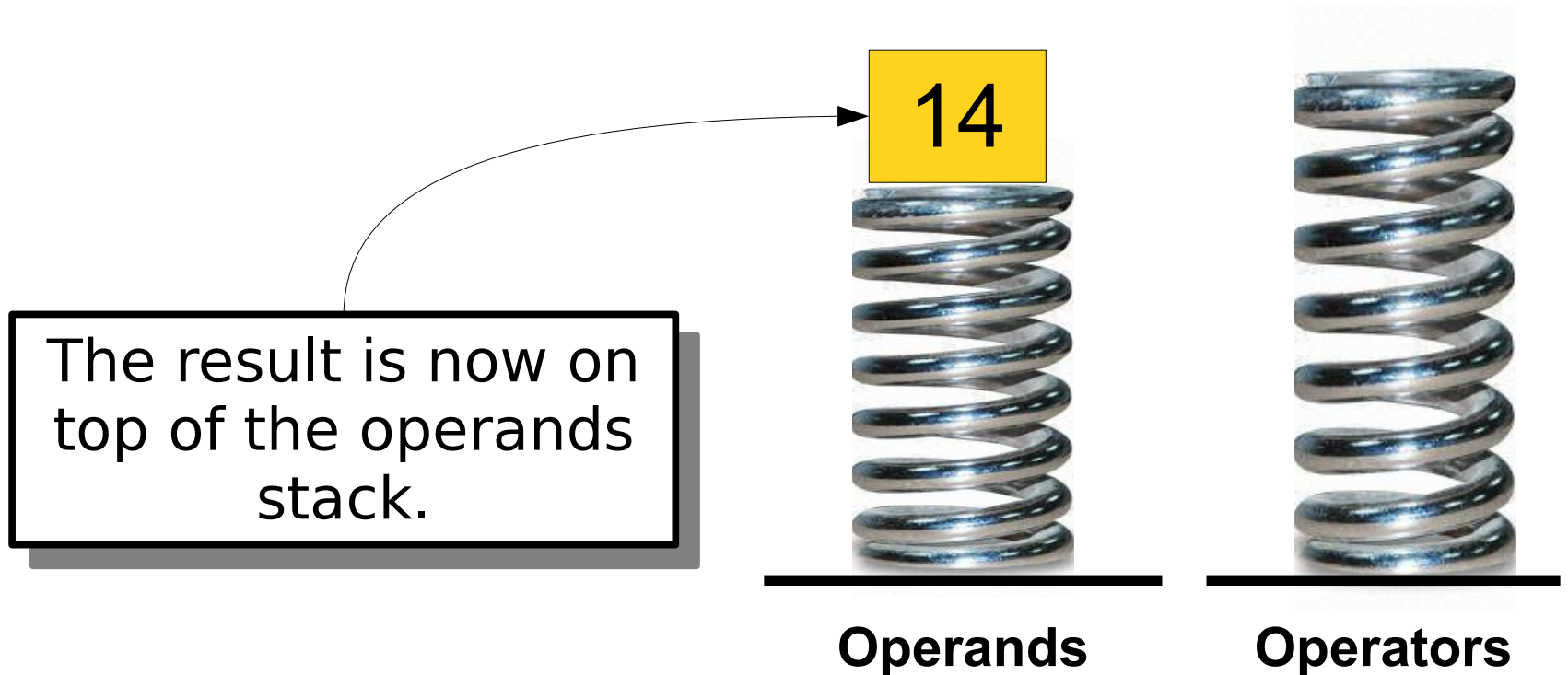
# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

2	+	3	*	5	-	6	/	2
---	---	---	---	---	---	---	---	---



# The Shunting-Yard Algorithm

- Maintain a stack of operators and a stack of operands.
- For each token:
  - If it's a number, push it onto the operand stack.
  - If it's an operator:
    - Keep evaluating operands until the scanned operator has higher precedence than the most recent operator.
    - Push the operator onto the operator stack.
- Once all input is done, keep evaluating operators until no operators remain.
- The value on the operand stack is the overall result.

Pseudo-code(On Board)

shunting-yard.cpp (Computer)

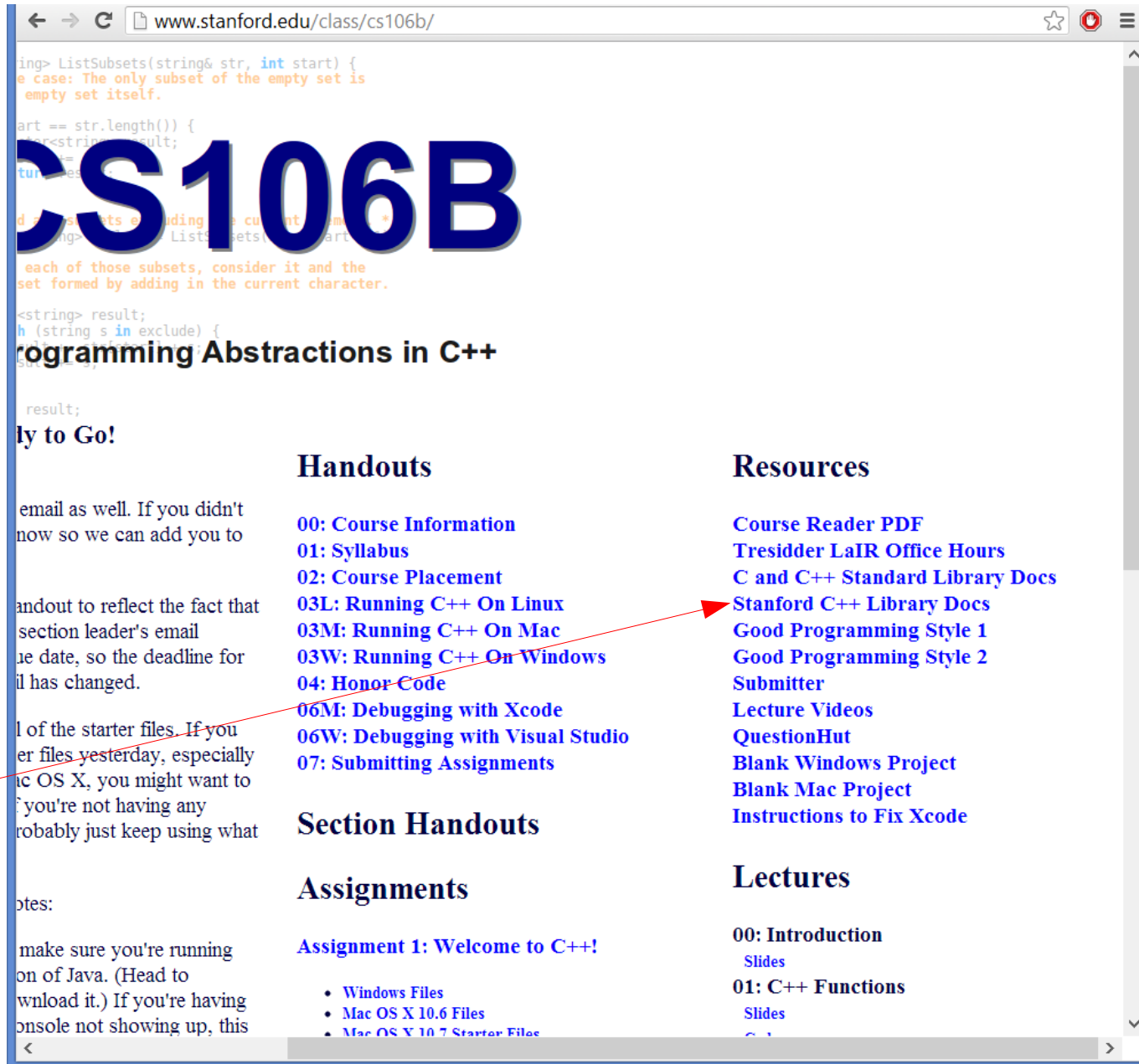
# Extensions to Shunting-Yard

- How might you update the shunting-yard algorithm to:
  - Handle/report syntax errors in the input?
  - Support parentheses?
  - Support functions like sin, cos, and tan?
  - Support variables?
- For more information on scanning and parsing, take **CS124** (*From Languages to Information*) or **CS143** (*Compilers*).

Hey Aubrey, do you expect me to memorize every  
method of every class?...

No! Computer Science is *not* about memorizing  
method names

# Collections Documentation



The screenshot shows a web browser window with the URL `www.stanford.edu/class/cs106b/`. The page features a large blue **CS106B** logo. Below the logo, there is a section titled **Programming Abstractions in C++**. The page is organized into several columns of links and text. A red arrow points from the left side of the page to the **Stanford C++ Library Docs** link in the Resources section.

```
ing> ListSubsets(string& str, int start) {
  // Base case: The only subset of the empty set is
  // the empty set itself.
  if (start == str.length()) {
    return <string> result;
  }
  // For each of those subsets, consider it and the
  // set formed by adding in the current character.
  <string> result;
  for (string s in exclude) {
    result;
  }
}
```

## Handouts

- [00: Course Information](#)
- [01: Syllabus](#)
- [02: Course Placement](#)
- [03L: Running C++ On Linux](#)
- [03M: Running C++ On Mac](#)
- [03W: Running C++ On Windows](#)
- [04: Honor Code](#)
- [06M: Debugging with Xcode](#)
- [06W: Debugging with Visual Studio](#)
- [07: Submitting Assignments](#)

## Resources

- [Course Reader PDF](#)
- [Tresidder LaIR Office Hours](#)
- [C and C++ Standard Library Docs](#)
- [Stanford C++ Library Docs](#)
- [Good Programming Style 1](#)
- [Good Programming Style 2](#)
- [Submitter](#)
- [Lecture Videos](#)
- [QuestionHut](#)
- [Blank Windows Project](#)
- [Blank Mac Project](#)
- [Instructions to Fix Xcode](#)

## Section Handouts

## Assignments

### Assignment 1: Welcome to C++!

- [Windows Files](#)
- [Mac OS X 10.6 Files](#)
- [Mac OS X 10.7 Starter Files](#)

## Lectures

- [00: Introduction](#)
  - [Slides](#)
- [01: C++ Functions](#)
  - [Slides](#)

# Next Time

- **Vector**
  - A standard collection for sequences.
- **Grid**
  - A standard collection for 2D data.