Collections, Part One
Announcements

• Assignment 1 (Welcome to C++!) due Monday, April 15 at 2:15PM.
  • Warm up with C++!
  • Play around with strings and recursion!
• Section assignments will be announced tomorrow. If you have not signed up for a section, visit the signup link tomorrow at 5PM:
  http://cs198.stanford.edu/section
• Mac users – if you're getting an error about “minimum deployment target,” we are looking into this and should get a fix posted to the course website soon. Our sincerest apologies!
Announcements

- Casual dinner for women studying CS this **Wednesday, April 10** at 5:00PM at the Gates Patio.
- Everyone is welcome!
One last C++ detail...
Reference Parameters

• In C++, *all* parameters are passed by value unless specified otherwise.
  • The parameter is initialized to a copy of the argument.
• You can pass a parameter by reference by annotating it with the & sign:
  
  ```cpp
  void removeSpaces(string& argument);
  void reverse(string& argument);
  ```
Yay! Now on to new things!
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.

- This week is dedicated to exploring different collections and how to harness them appropriately.

- We'll discuss efficiency issues and implementations later on.
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

- 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
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
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
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
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
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

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

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

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

```c
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

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

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

Balancing Parentheses

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

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

^
Balancing Parentheses

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

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

Balancing Parentheses

\[
\text{int foo()} \{ \text{if (x * (y + z[1]) < 137) \{ x = 1; \}} \}
\]
Balancing Parentheses

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

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

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

Balancing Parentheses
Balancing Parentheses

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

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

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

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

Balancing Parentheses

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

^
Balancing Parentheses

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

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

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

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

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

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

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

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

Balancing Parentheses

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

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

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

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

[Diagram of balanced parentheses]
Balancing Parentheses

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

![Diagram of balanced parentheses]
Balancing Parentheses

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

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

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

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

Balancing Parentheses

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

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

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

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

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

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

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

Interesting exercise: code this up!
Application: Evaluating Expressions
Evaluating Expressions

- Evaluating expressions is much trickier than it might seem due to issues of precedence.
  - $1 + 3 \times 5 - 7 = 9$
  - $4 / 2 + 2 = 4$
  - $17 \% 6 \% 3 = 2$

- 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.

- For now, let's assume we have a scanner. How might we handle parsing?
The Shunting-Yard Algorithm

\[
\begin{array}{c|c|c|c|c|c|c}
2 & + & 3 & * & 5 & - & 6 & \div & 2 \\
\end{array}
\]
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

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
</table>

Operands

Operators
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>+</td>
<td>3</td>
<td>*</td>
<td>5</td>
<td>-</td>
<td>6</td>
<td>/</td>
<td>2</td>
</tr>
</tbody>
</table>

Operands

Operators
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

+ 3 * 5 - 6 / 2

Operands

Operators
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>3</td>
<td>*</td>
<td>5</td>
<td>-</td>
<td>6</td>
<td>/</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Operands

Operators
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

+ 3 * 5 - 6 / 2

Operands

Operators
# The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>*</td>
<td>5</td>
<td>-</td>
<td>6</td>
<td>/</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Operands

Operators

<table>
<thead>
<tr>
<th>2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>+</th>
</tr>
</thead>
</table>
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

3 * 5 - 6 / 2

Operands

Operators

2

+
The Shunting-Yard Algorithm

Operands:

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

Operators:

- 3
- *
- 5
- -
- 6
- /
- 2
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

* 5 - 6 / 2

Operands

Operators
The Shunting-Yard Algorithm

\[
\begin{array}{cccccc}
2 & + & 3 & * & 5 & - & 6 & / & 2 \\
\end{array}
\]

**Operands**

\[
\begin{array}{cccccc}
* & 5 & - & 6 & / & 2 \\
\end{array}
\]

**Operators**
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

* 5 - 6 / 2

3
2
+

Operands

Operators
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>5</td>
<td>-</td>
<td>6</td>
<td>/</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Operands

Operators
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

Operands

Operators

3

2

* +
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

5 - 6 / 2

Operands

3
2

Operators

* +
## The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
</table>

| 5 | - | 6 | / | 2 |

**Operands**

![Operands Diagram]

**Operators**

![Operators Diagram]
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
</table>

- 6 | / | 2 |

Operands

Operators
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

- 6 / 2

Operands

Operators

5
3
2

* +
# The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
</table>

| - | 6 | / | 2 |

**Operands**

- 2
- 3
- 5
- 6
- 2

**Operators**

- +
- *
- /
The Shunting-Yard Algorithm
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

Subtraction has lower precedence than multiplication, so we need to evaluate the multiply before the subtract.
The Shunting-Yard Algorithm

Operands

2 + 3 * 5 - 6 / 2

Operators

- 6 / 2

5

3

2

* +

Operands

Operators
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

- 6 / 2

3 * 5

2 +

Operands

Operators
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>6</td>
<td>/</td>
<td>2</td>
</tr>
</tbody>
</table>

- 6 / 2
- 15

Operands

Operators
The Shunting-Yard Algorithm

Operands

2 + 3 * 5 - 6 / 2

Operators

- 6 / 2

15
2

+
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

- 6 / 2

15

2

+
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

15
2

+
Subtraction has equal precedence to addition. Since addition is left-associative, we evaluate the add before the subtract.
The Shunting-Yard Algorithm

\[
\begin{array}{cccccc}
2 & + & 3 & * & 5 & - & 6 & / & 2 \\
\end{array}
\]

\[
\begin{array}{cccc}
- & 6 & / & 2 \\
\end{array}
\]

Operands

15

2

Operators

+
The Shunting-Yard Algorithm

Operands: 2, 3, 5, 6, 2
Operators: +, *, -, /, 15

2 + 3 * 5 - 6 / 2

- 6 / 2

2 + 15
The Shunting-Yard Algorithm

Operands:

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

Operators:

| - | 6 | / | 2 |

Result: 17
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>Operands</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>/</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
</tr>
</tbody>
</table>

17

Operands

Operators
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>6</td>
<td>/</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

17

Operands

Operators
The Shunting-Yard Algorithm

\[
\begin{array}{ccccccccc}
2 & + & 3 & * & 5 & - & 6 & / & 2 \\
\end{array}
\]

\[
\begin{array}{cccccc}
6 & / & 2 \\
\end{array}
\]
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

6 / 2

17 -
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

17 -

Operands
Operators
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>Operands</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>+</td>
</tr>
</tbody>
</table>

| 6 | 17 | - | 2 |
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

Operands: 2, 3, 5, 6, 2
Operators: +, *, -, /
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

Operands

Operators

6
17

/ 2

-
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

Operands:

Operators:

2

6
17

/ 
-

Operands

Operators
The Shunting-Yard Algorithm

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

Operands

Operators

2

6

17

/ 

-
The Shunting-Yard Algorithm

Operands

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

Operators

- 2
- 6
- 17
- /
The Shunting-Yard Algorithm

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

Operands

- 2
- 6
- 17

Operators

- /
- -
The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>Operands</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>/</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Now that we've read all the tokens, we can finish evaluating all the expressions.
The Shunting-Yard Algorithm

2  +  3  *  5  -  6  /  2

Operands

Operators
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

17

6 / 2

-
The Shunting-Yard Algorithm

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

Operands: 2, 3, 5, 6, 2
Operators: +, *, -, /

17
3
-
The Shunting-Yard Algorithm

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

<table>
<thead>
<tr>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
</tr>
</tbody>
</table>

Operands

Operators

-
The Shunting-Yard Algorithm

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>+</td>
<td>3</td>
<td>*</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

Operands

3
17

Operators

-
The Shunting-Yard Algorithm

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

Operands

Operators
The Shunting-Yard Algorithm

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

17 - 3
The Shunting-Yard Algorithm

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

14

Operands

Operators
## The Shunting-Yard Algorithm

<table>
<thead>
<tr>
<th>2</th>
<th>+</th>
<th>3</th>
<th>*</th>
<th>5</th>
<th>-</th>
<th>6</th>
<th>/</th>
<th>2</th>
</tr>
</thead>
</table>

Operands

Operators
The Shunting-Yard Algorithm

\[
\begin{array}{cccccccc}
2 & + & 3 & * & 5 & - & 6 & / & 2 \\
\end{array}
\]

Operands

Operators

\[14\]
The Shunting-Yard Algorithm

2 + 3 * 5 - 6 / 2

The result is now on top of the operands stack.
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 current 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.
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**!
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).
Next Time

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