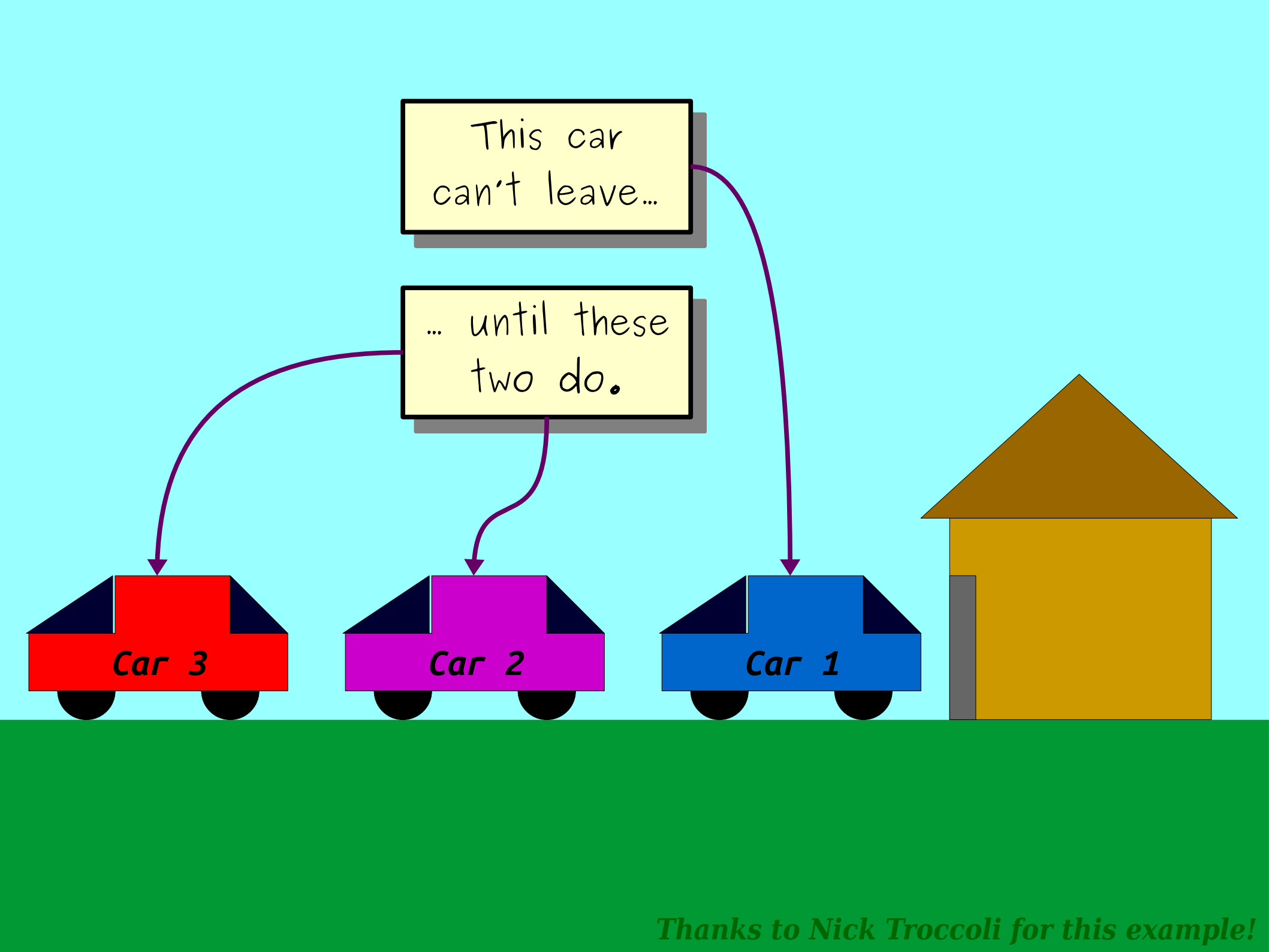
Containers

Part Three

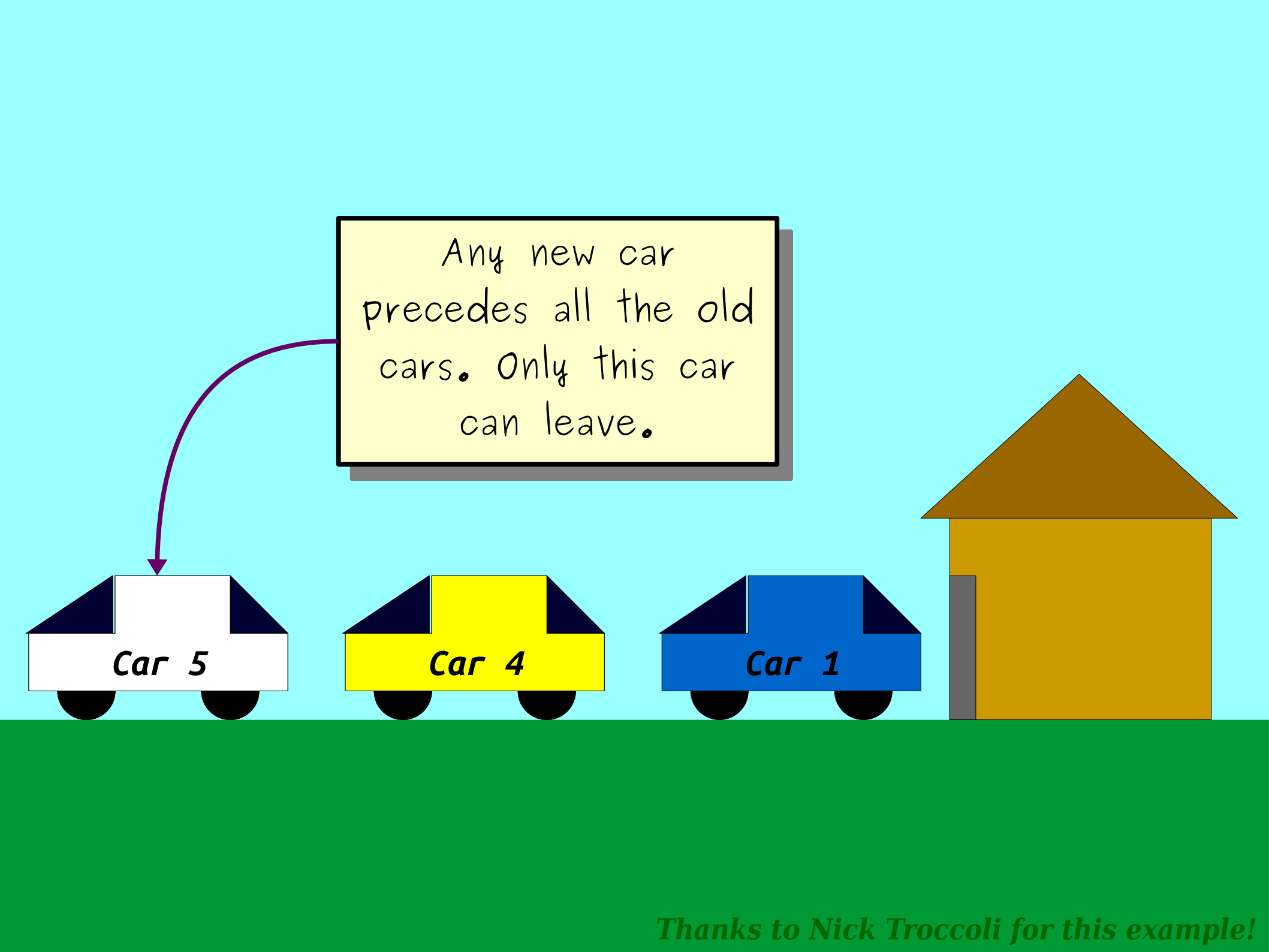
Outline for Today

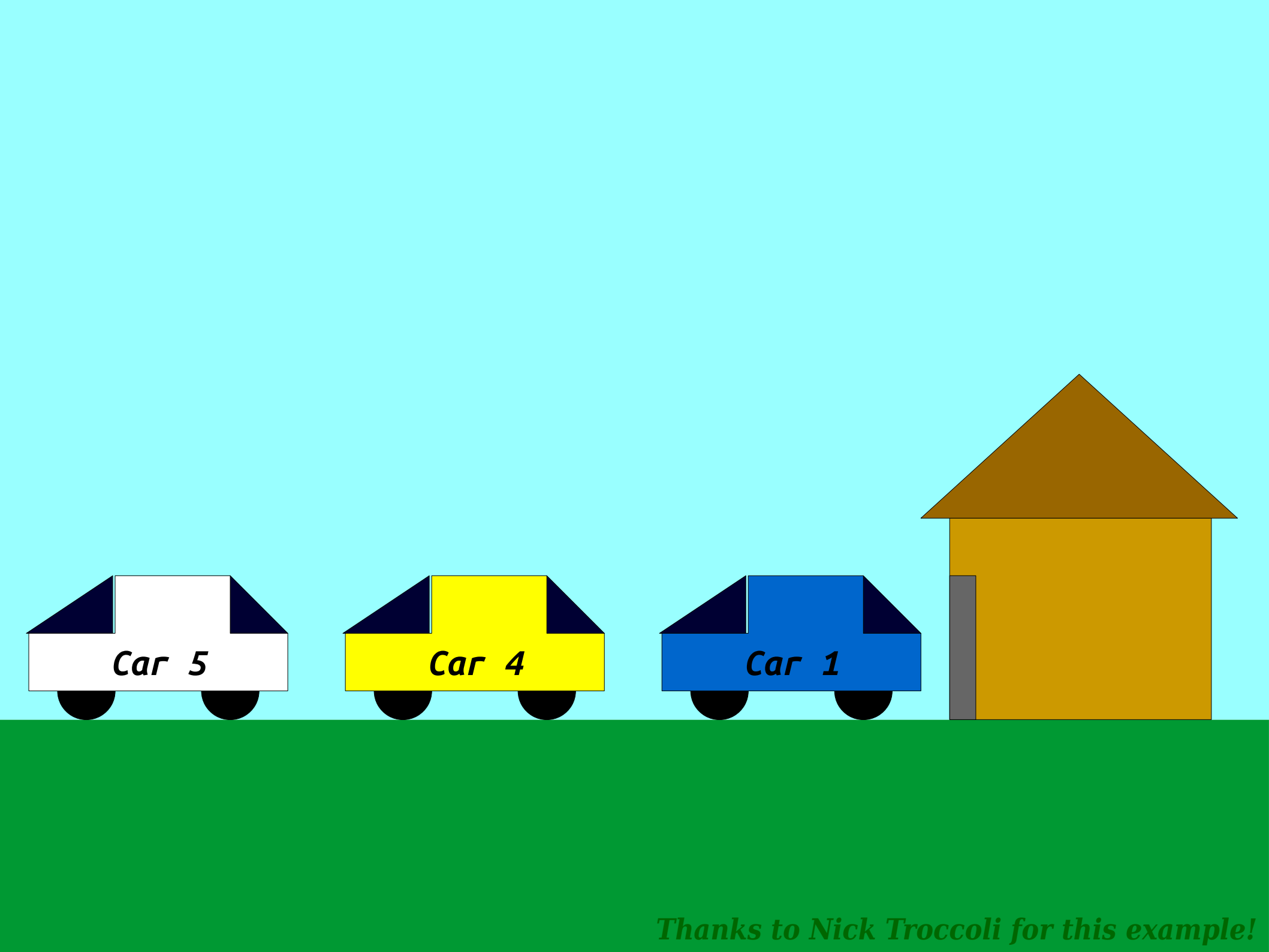
- ***Stacks***
  - Pancakes meets parsing!
- ***Queues***
  - Playing some music!

Stack



Thanks to Nick Troccoli for this example!


Any new car precedes all the old cars. Only this car can leave.
Car 5
Car 4
Car 1
Thanks to Nick Troccoli for this example!


Car 5
Car 4
Car 1
Thanks to Nick Troccoli for this example!

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.

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.

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.

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.

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.

Stack

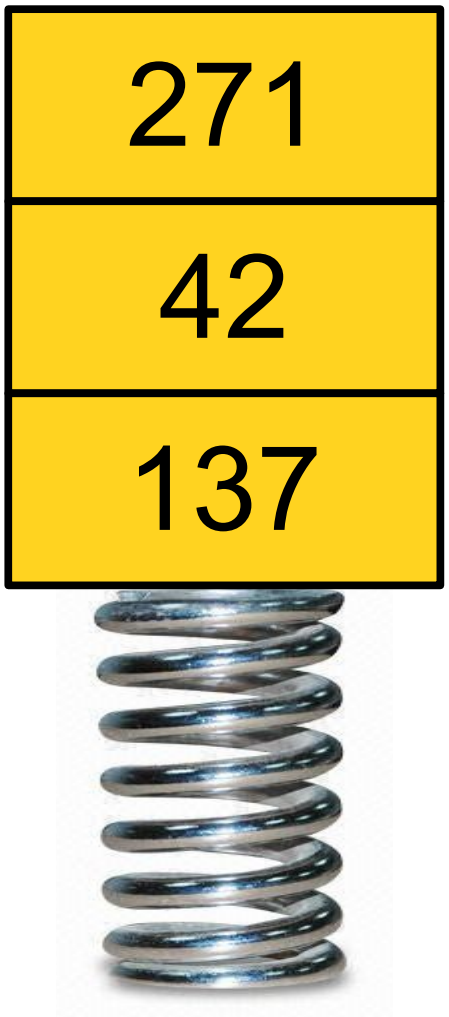
- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

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.

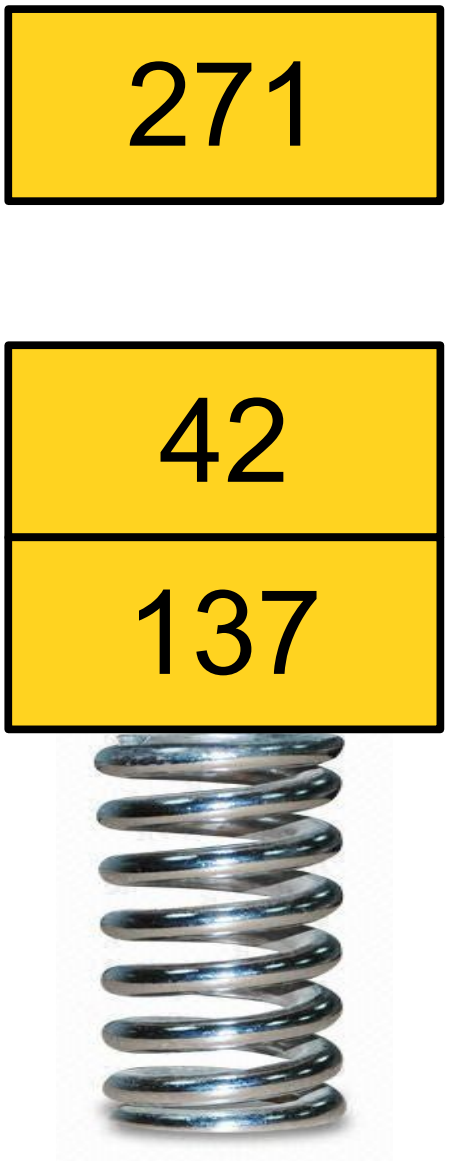
Stack

- 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.



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.



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.

Stack

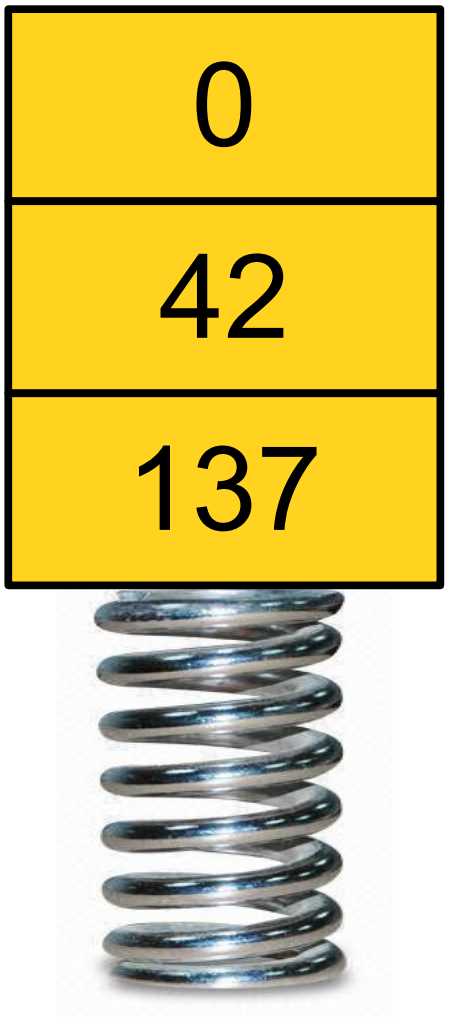
- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

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.

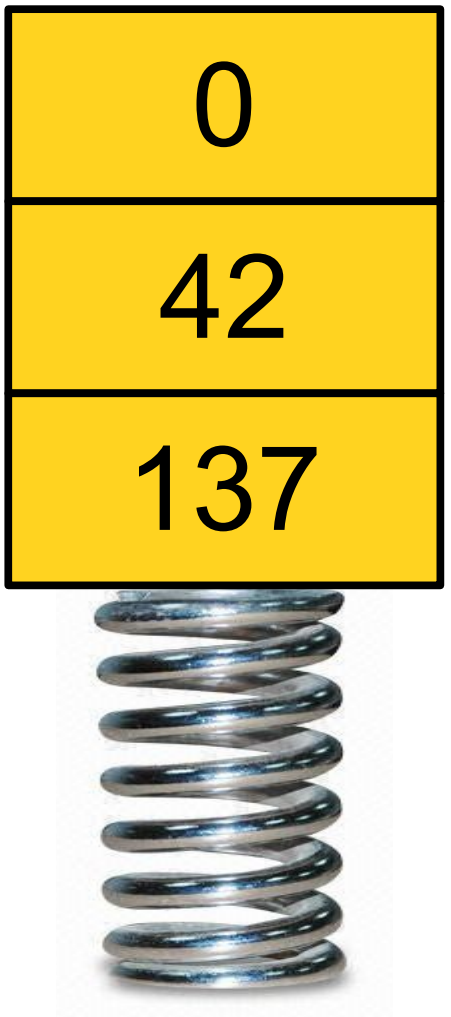
Stack

- 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.



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 topmost element of a `Stack` can be accessed.
- Do you see why we call it the *call stack* and talk about *stack frames*?



Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

Answer at
https://cs106b.stanford.edu/pollev

Stack

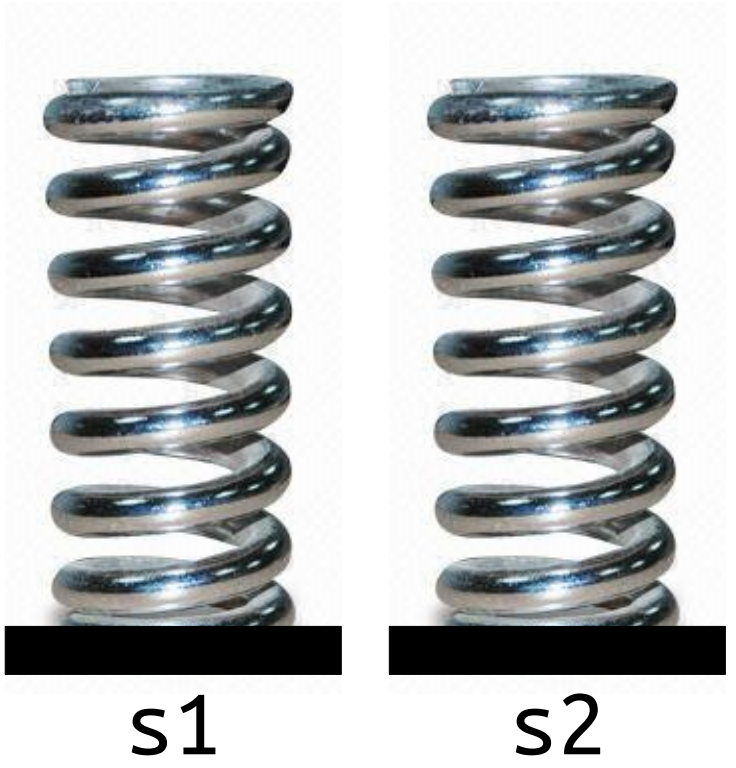
What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

Stack

What does this code print?

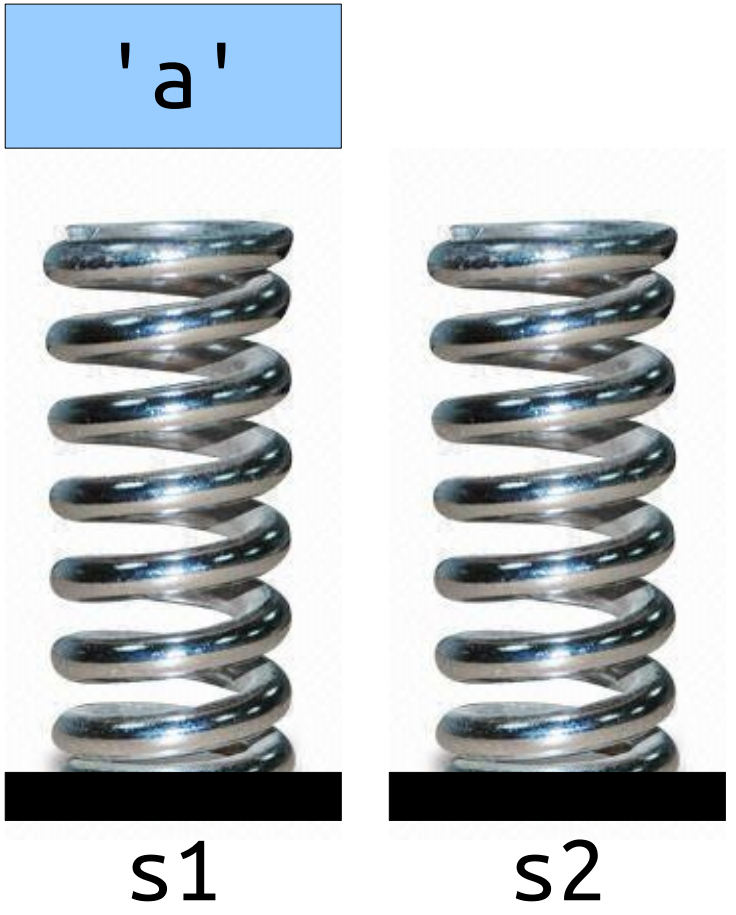
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

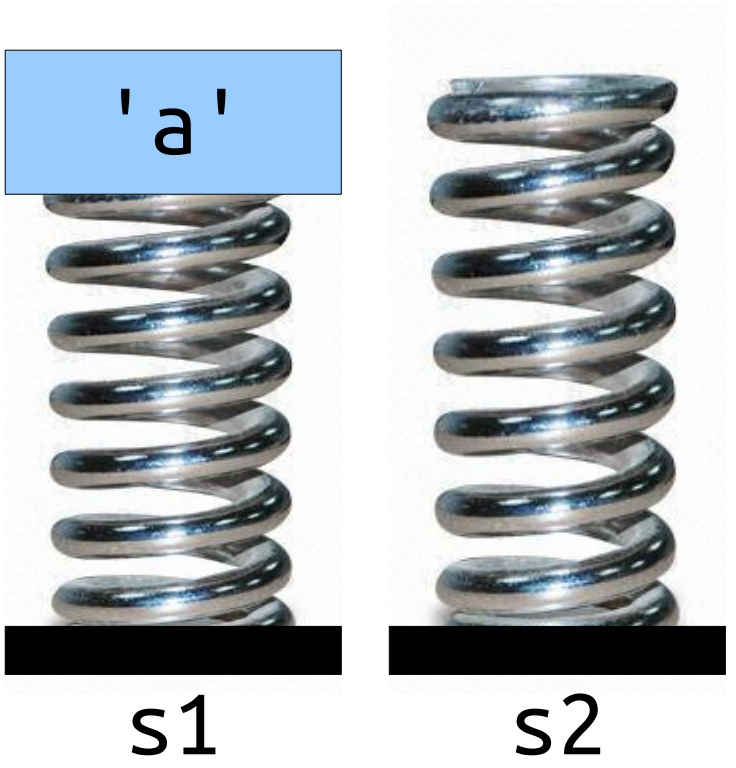
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

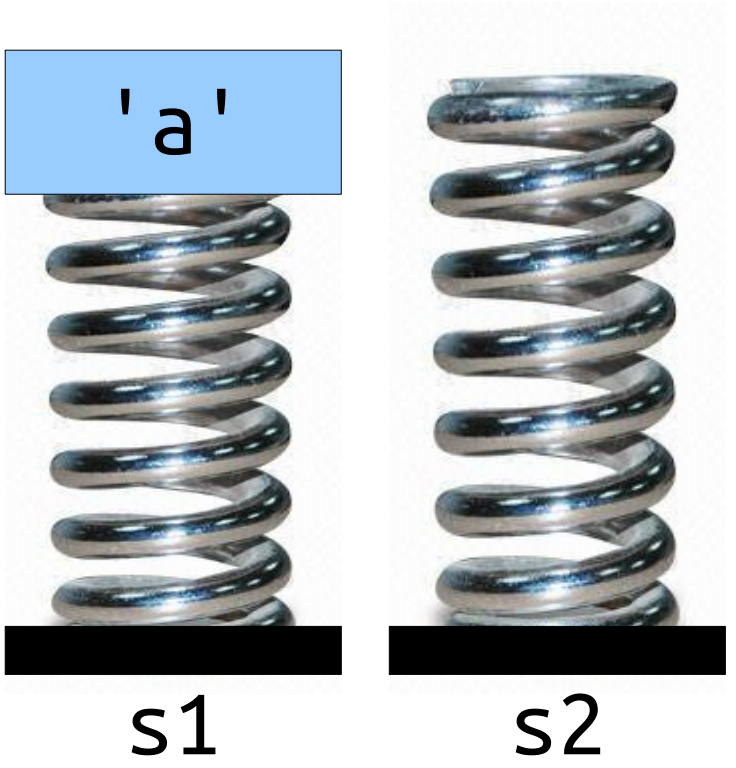
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

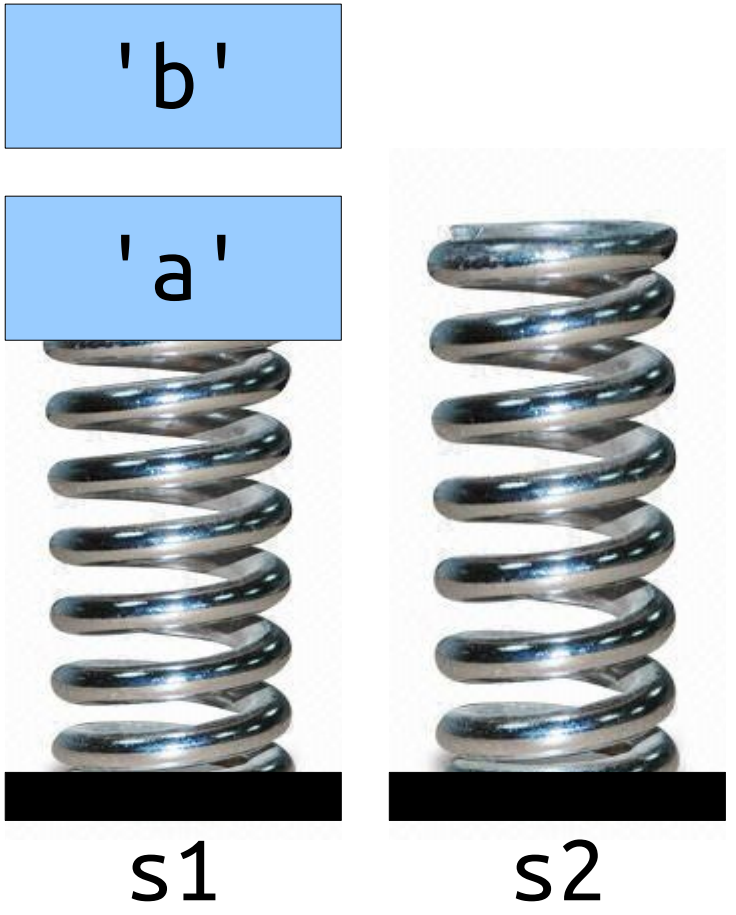
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

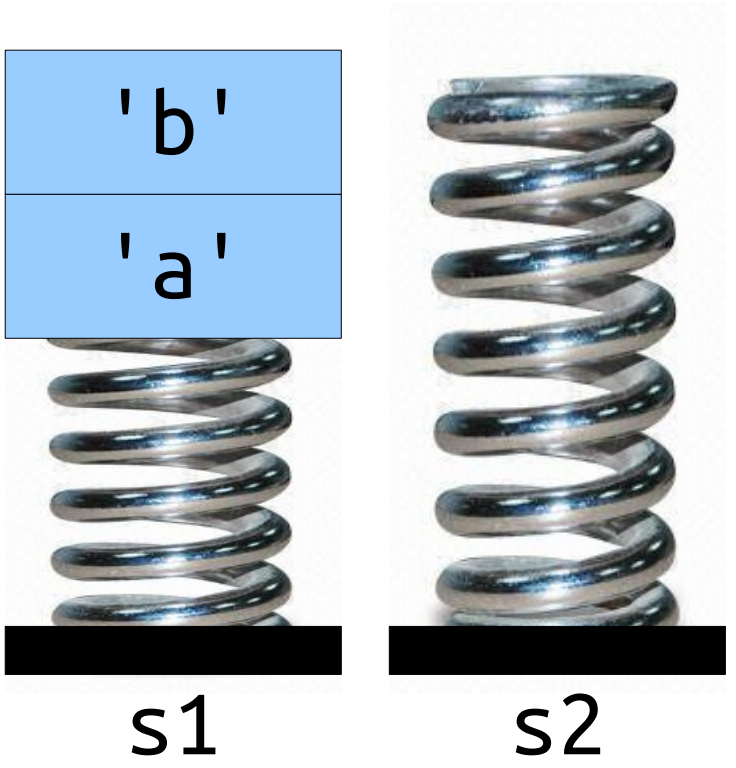
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

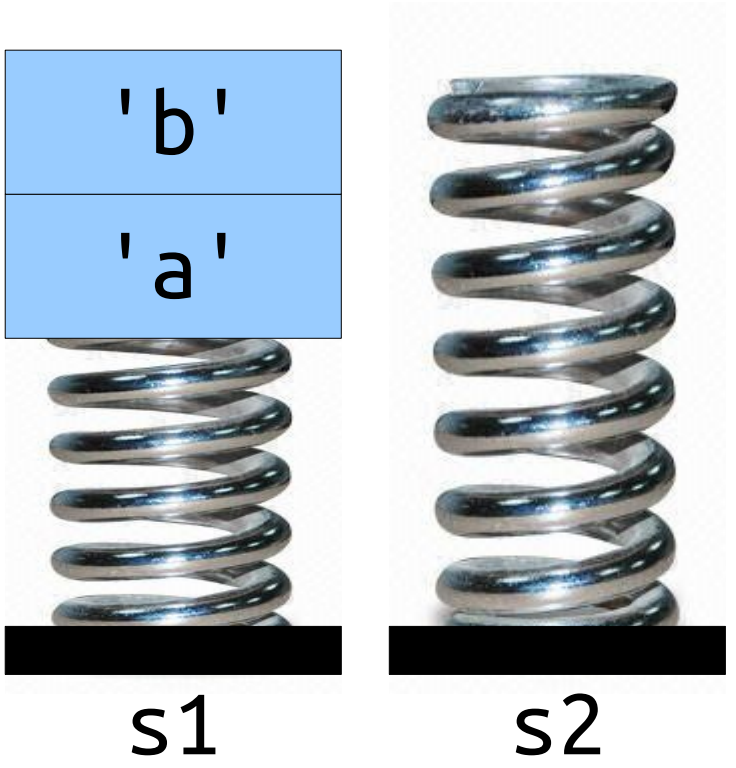
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

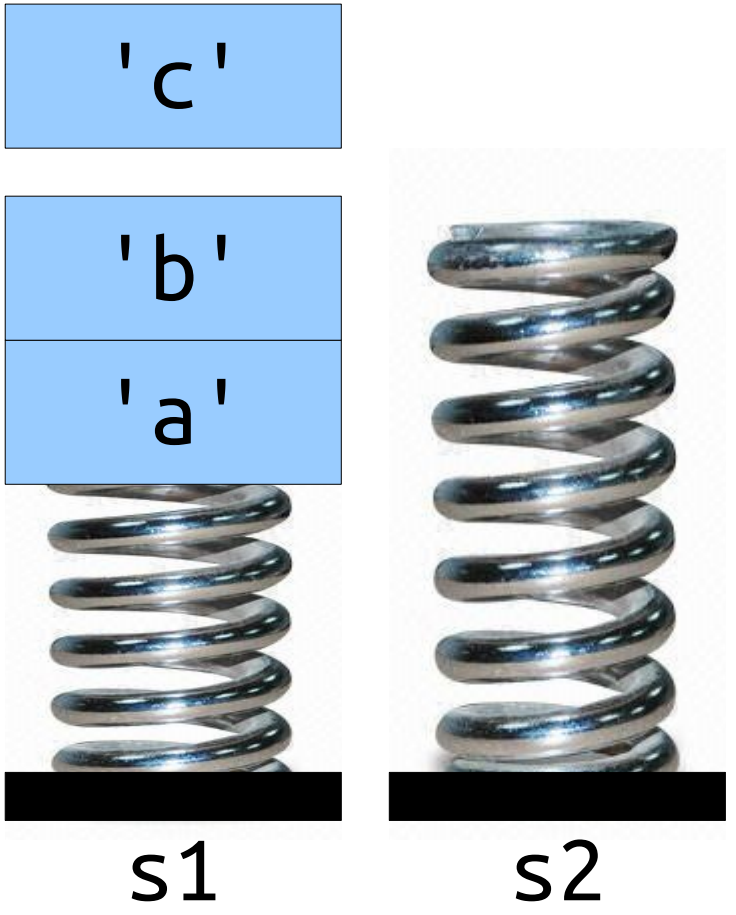
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

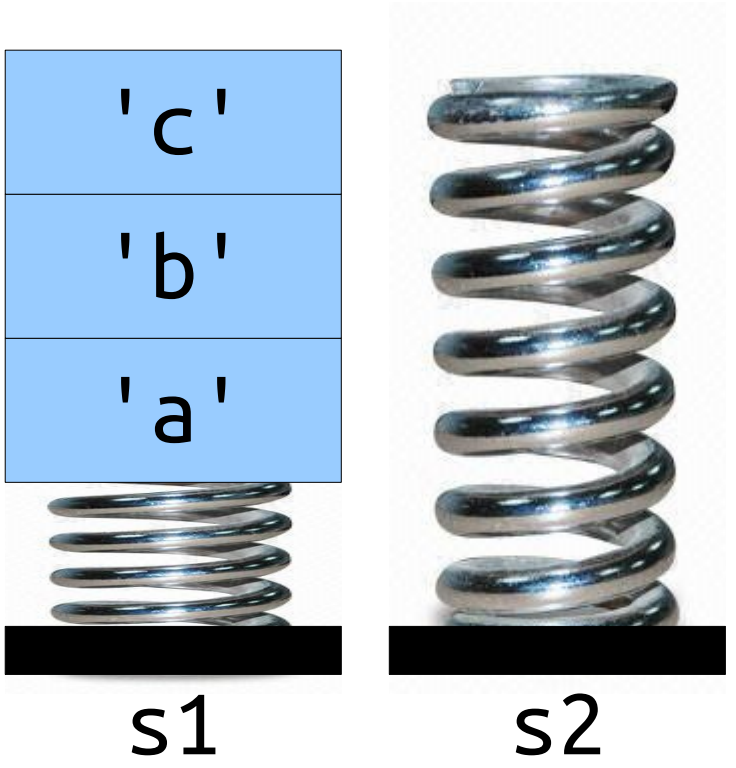
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

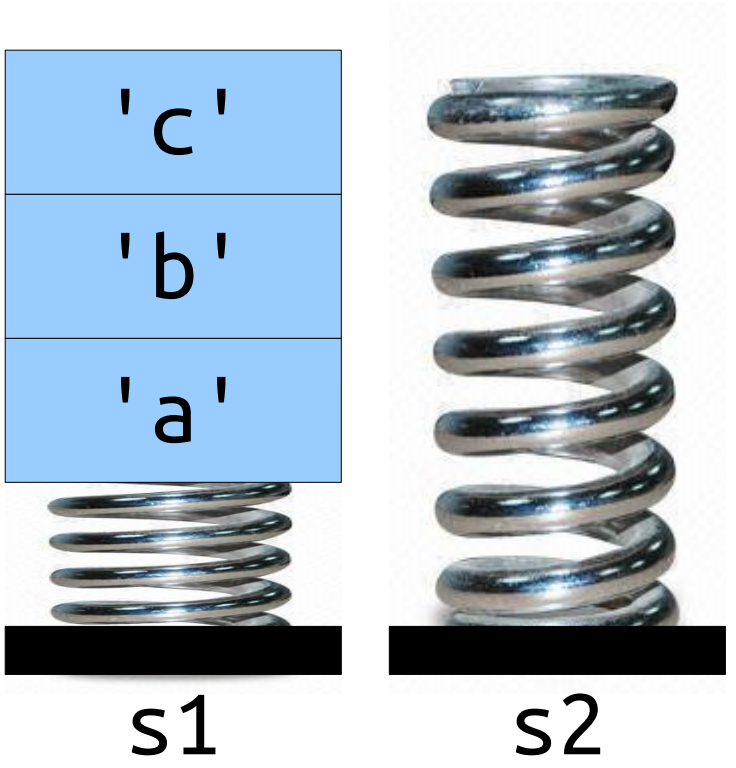
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

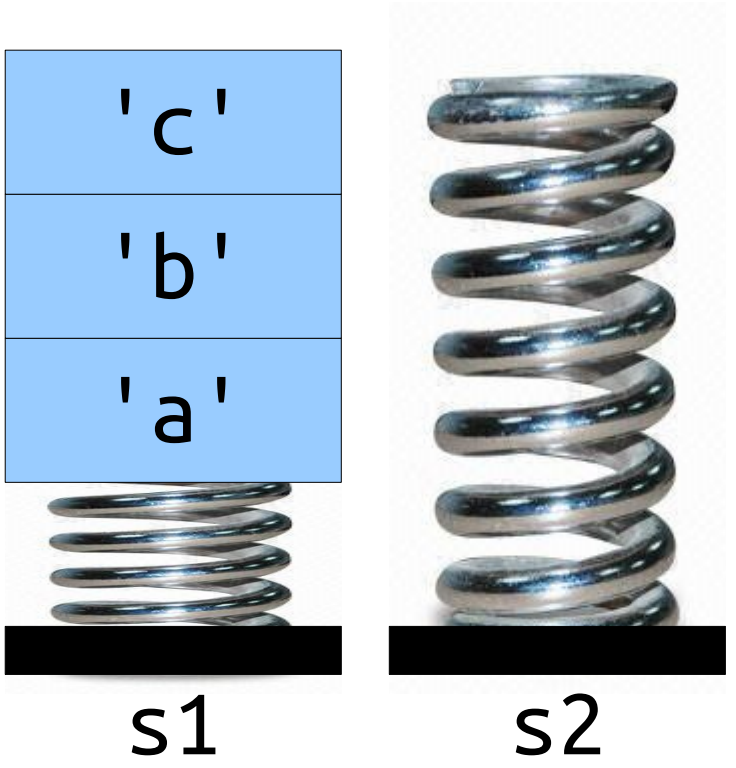
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

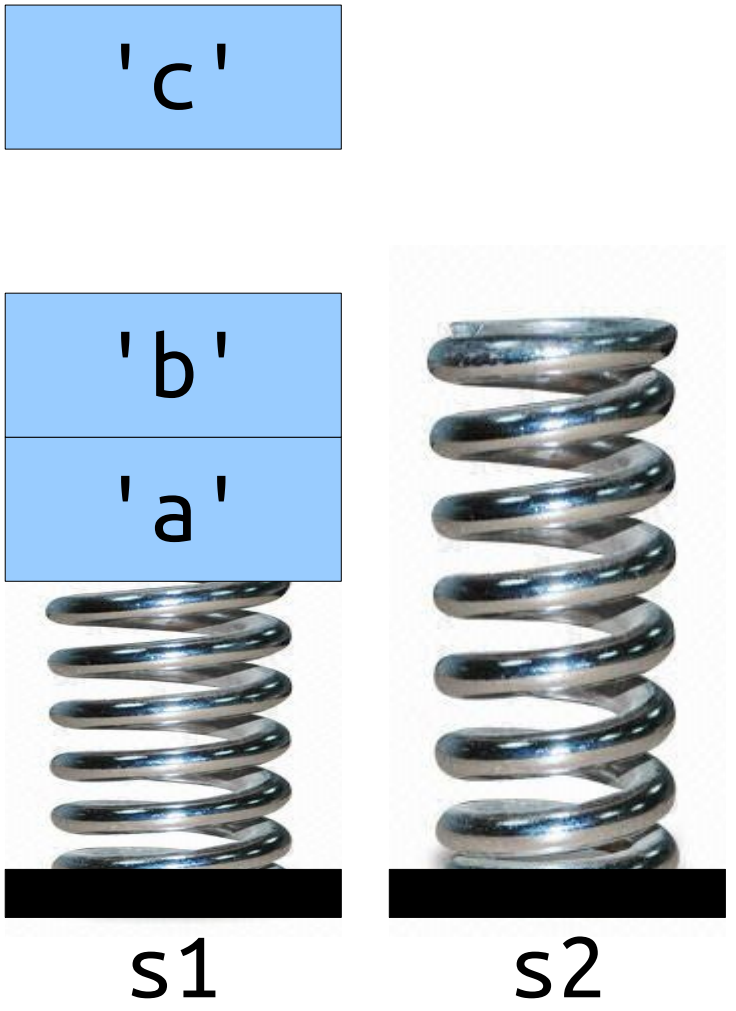
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

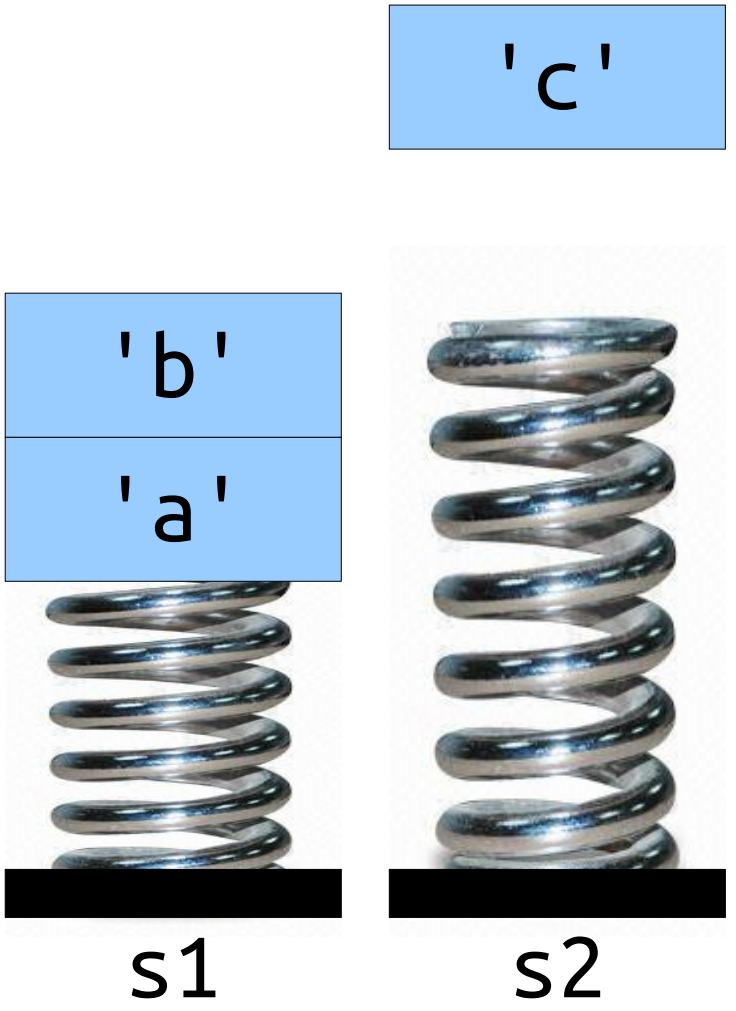
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

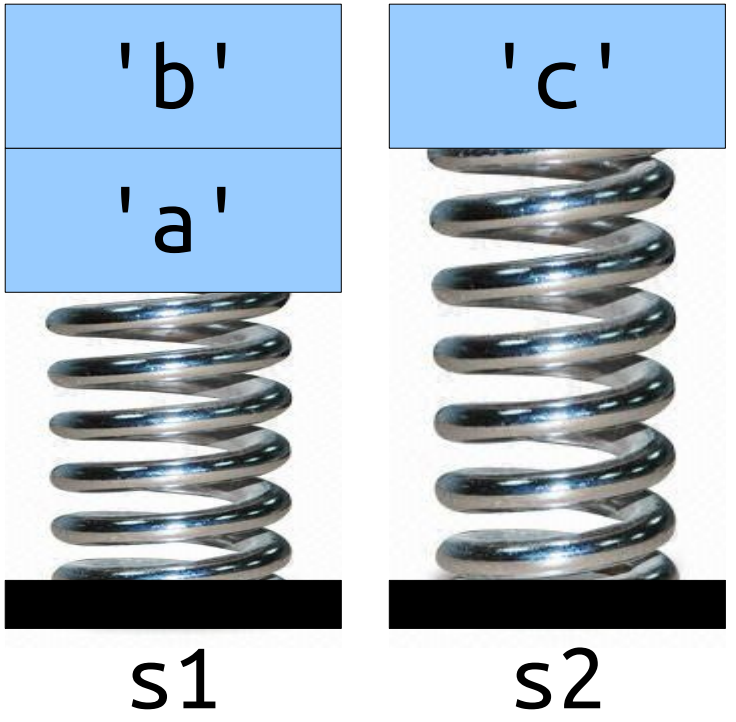
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

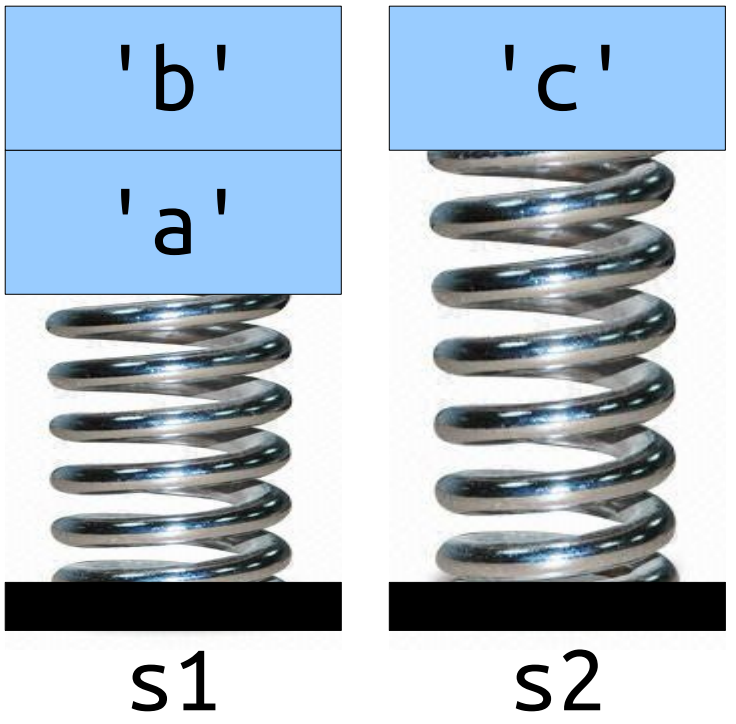
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

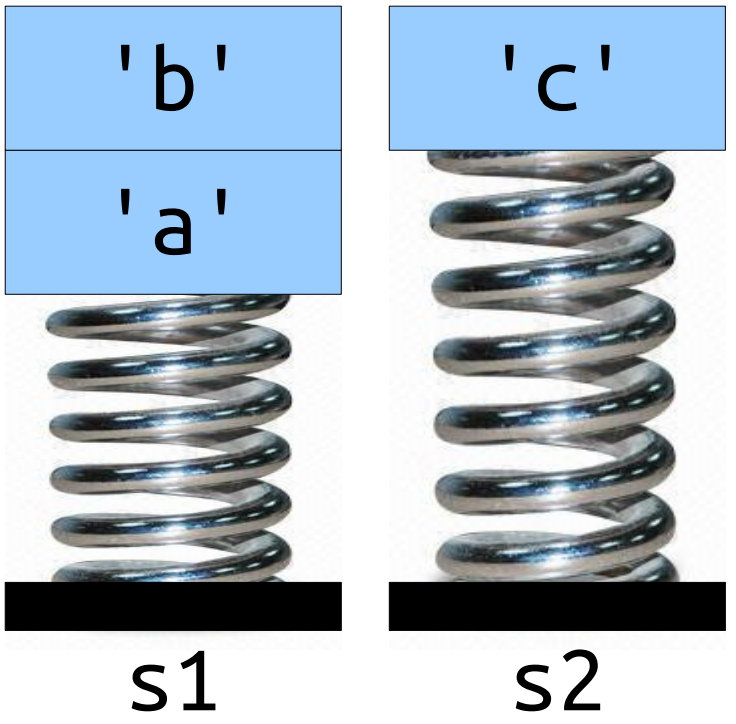
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

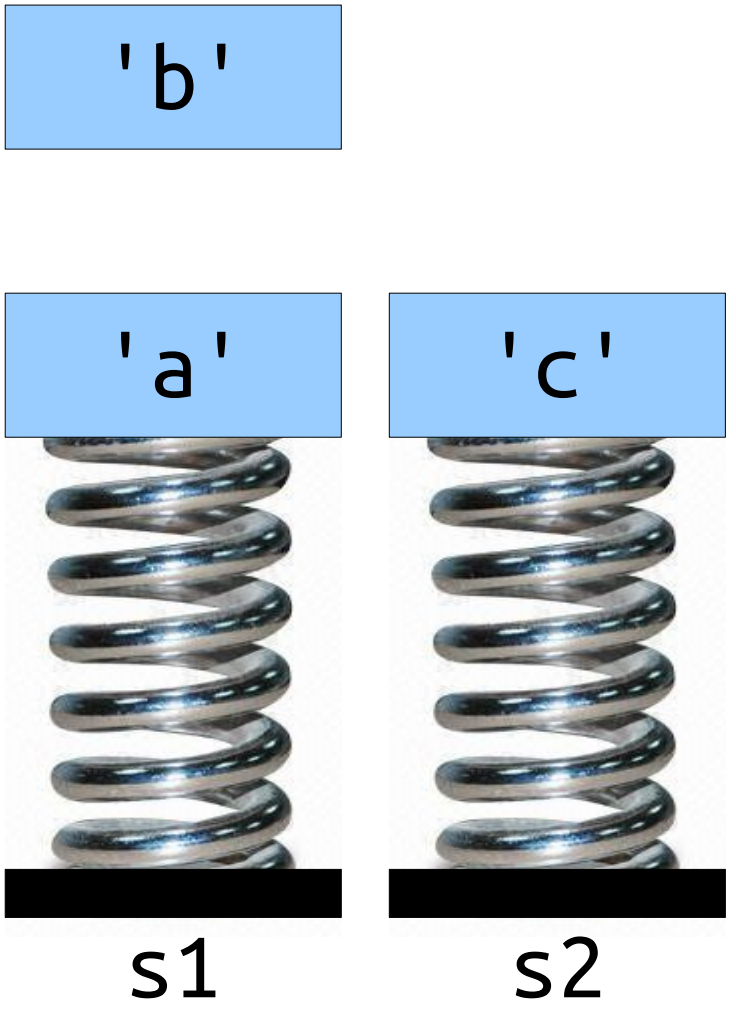
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

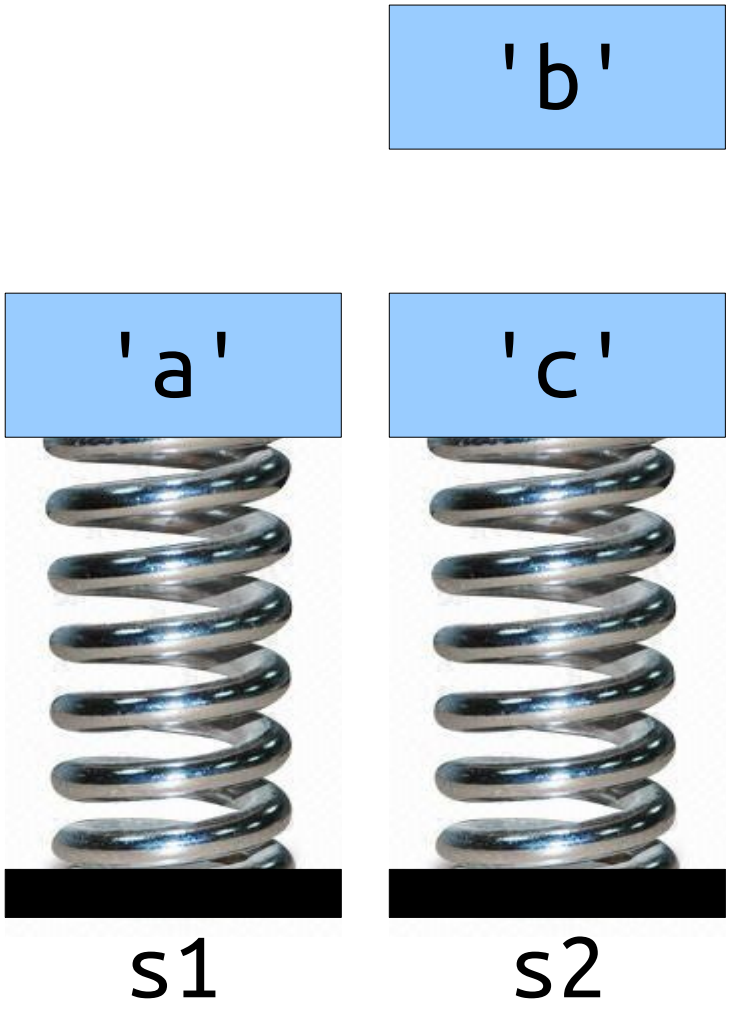
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

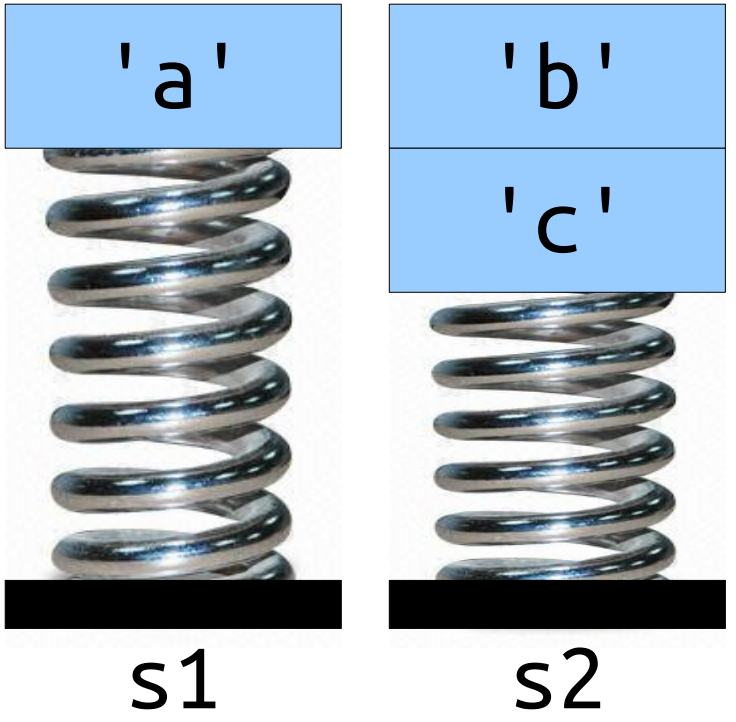
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

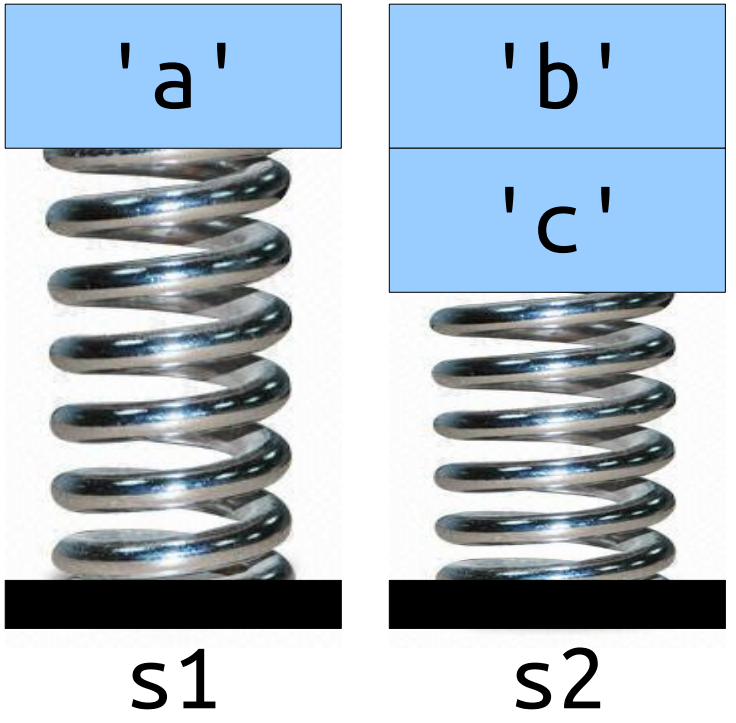
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

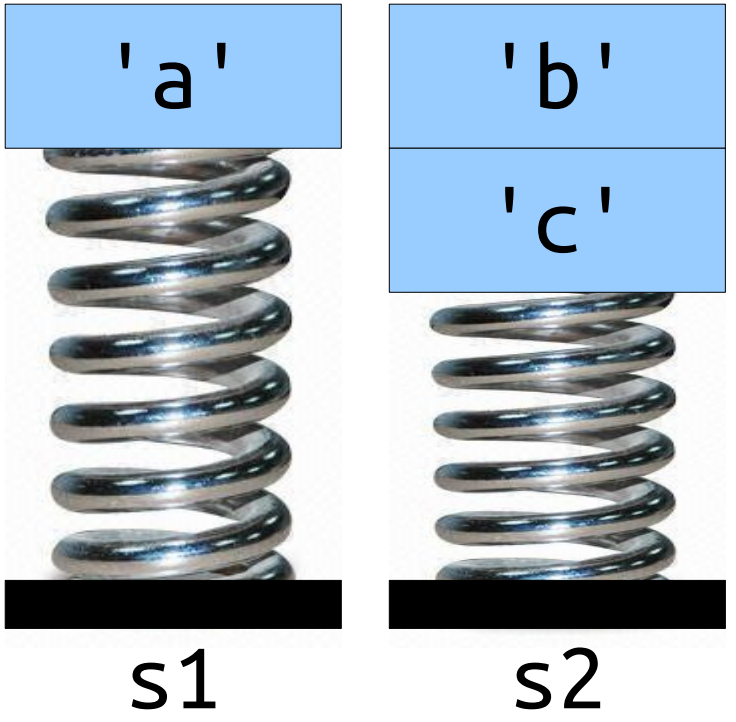
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

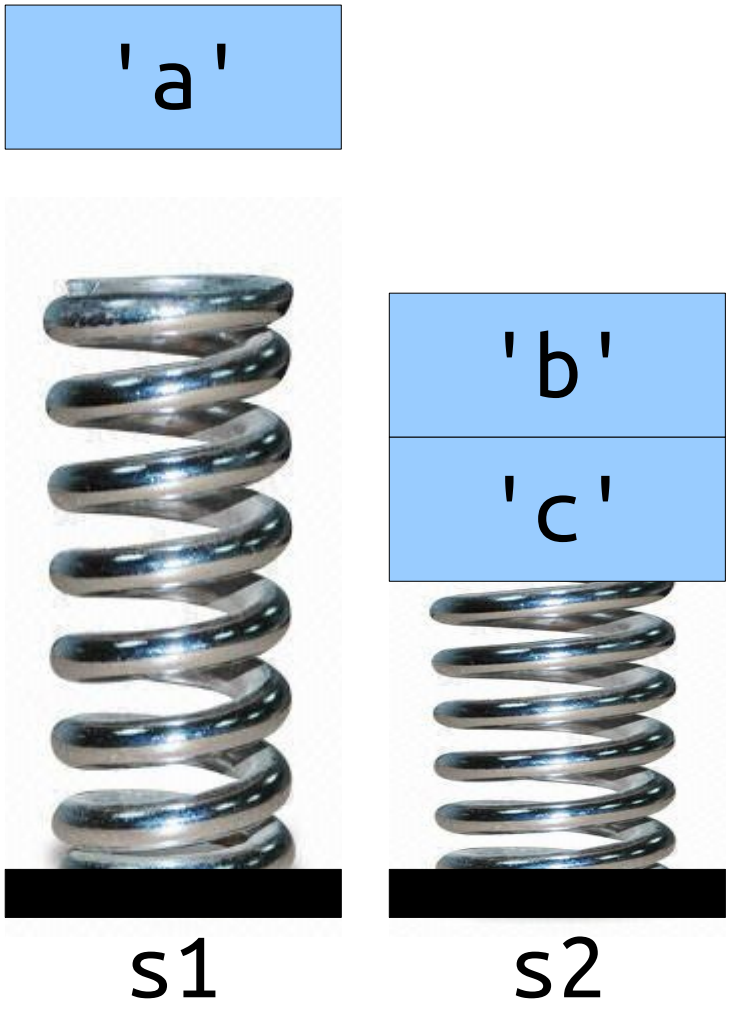
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

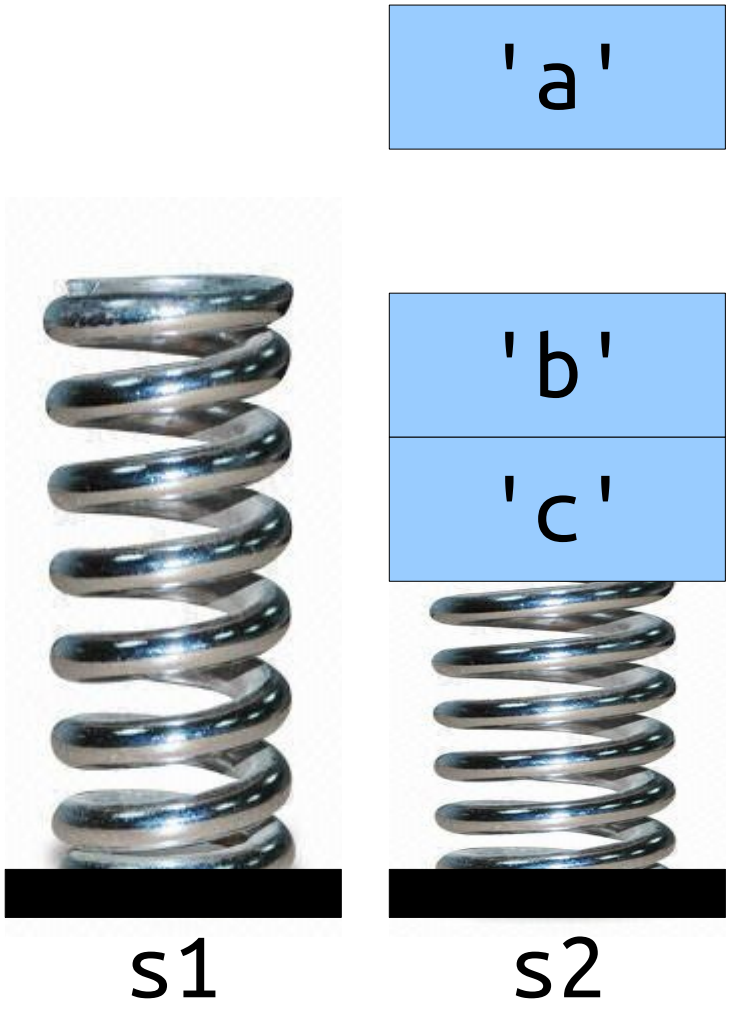
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

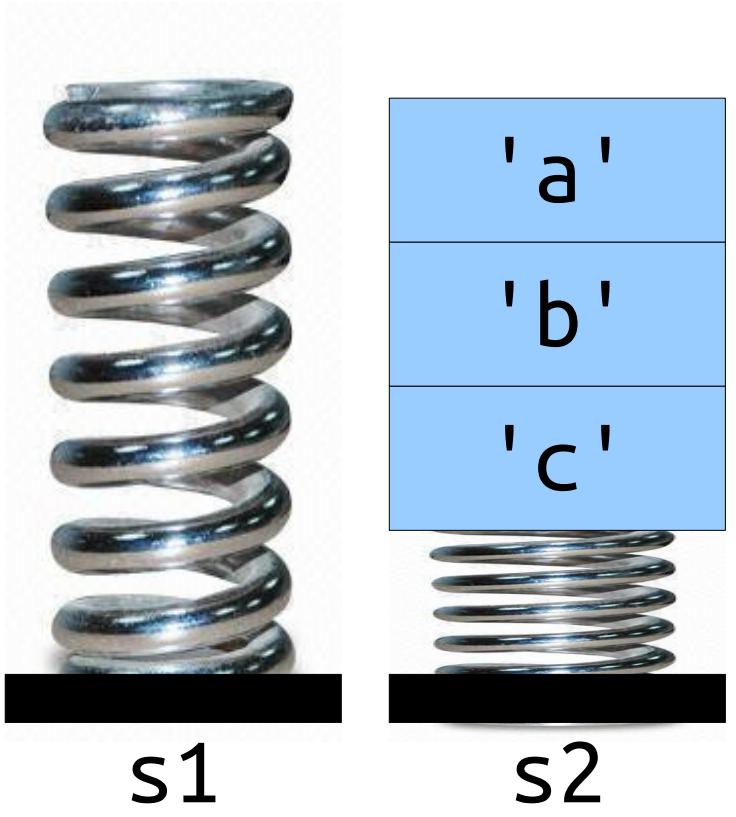
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

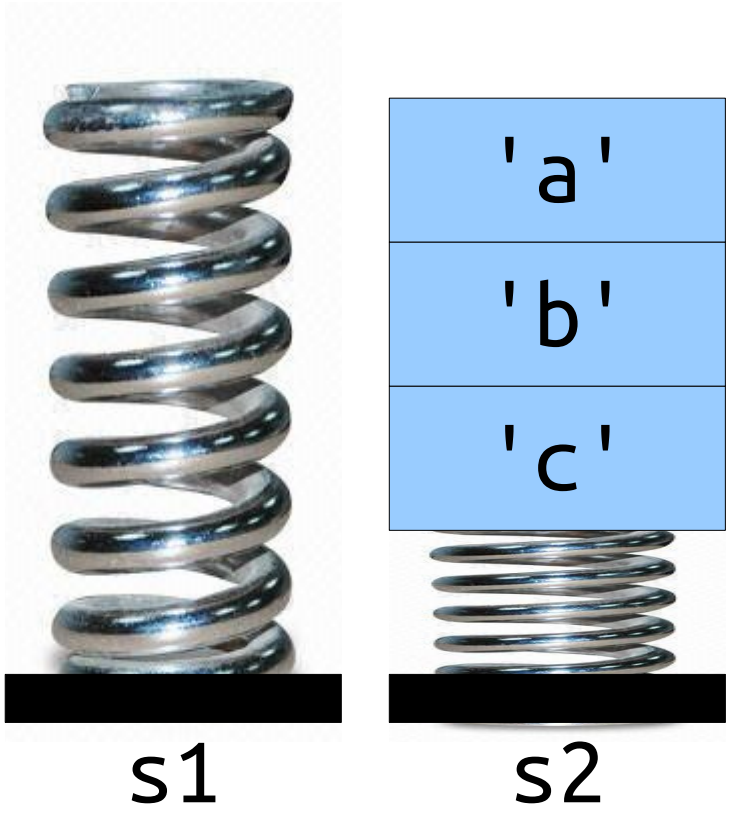
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

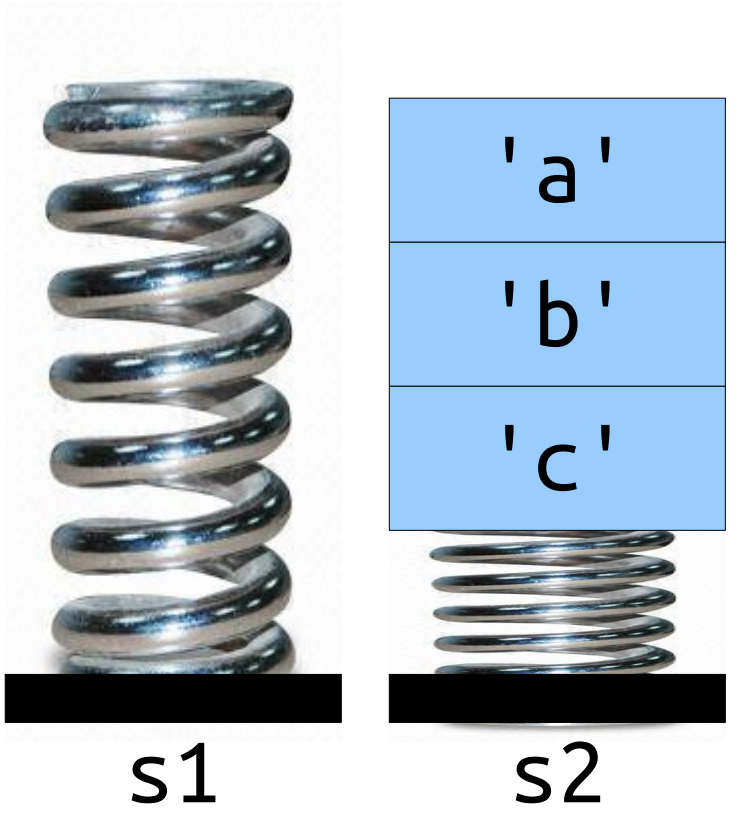
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

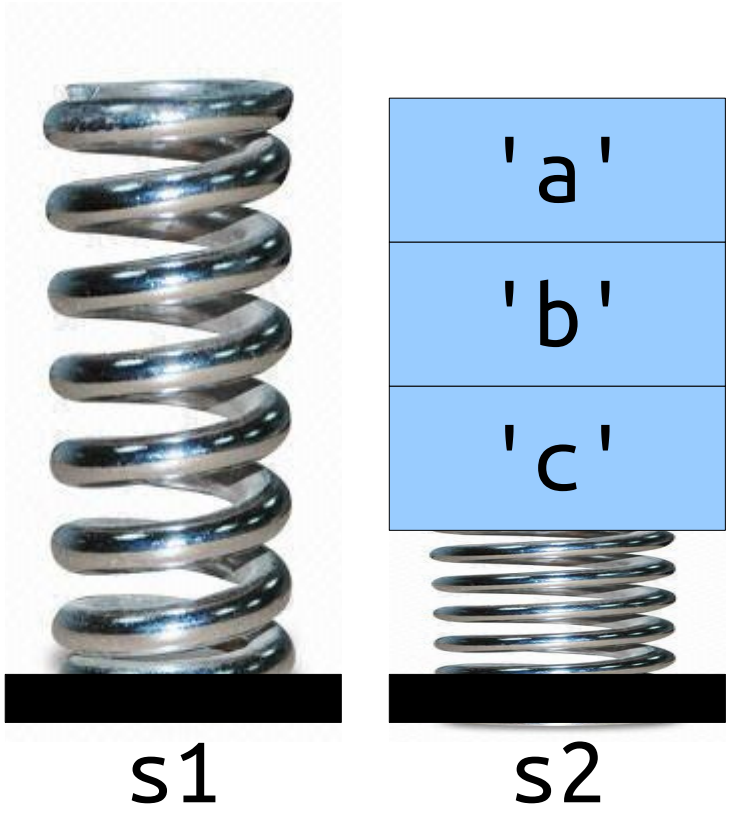
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

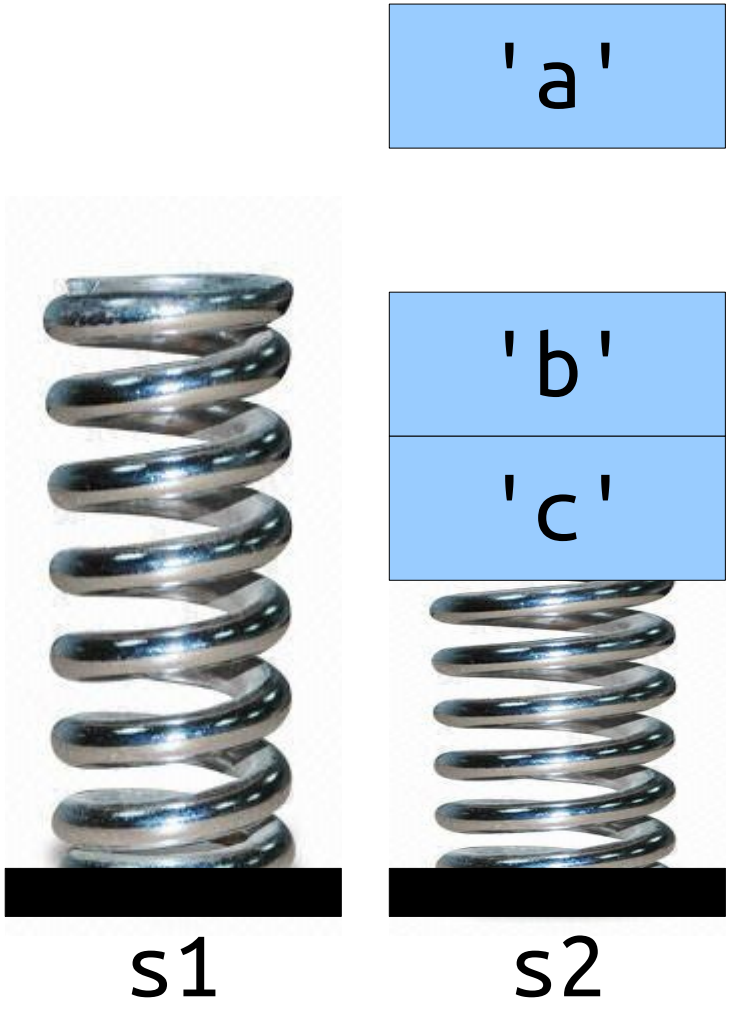
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

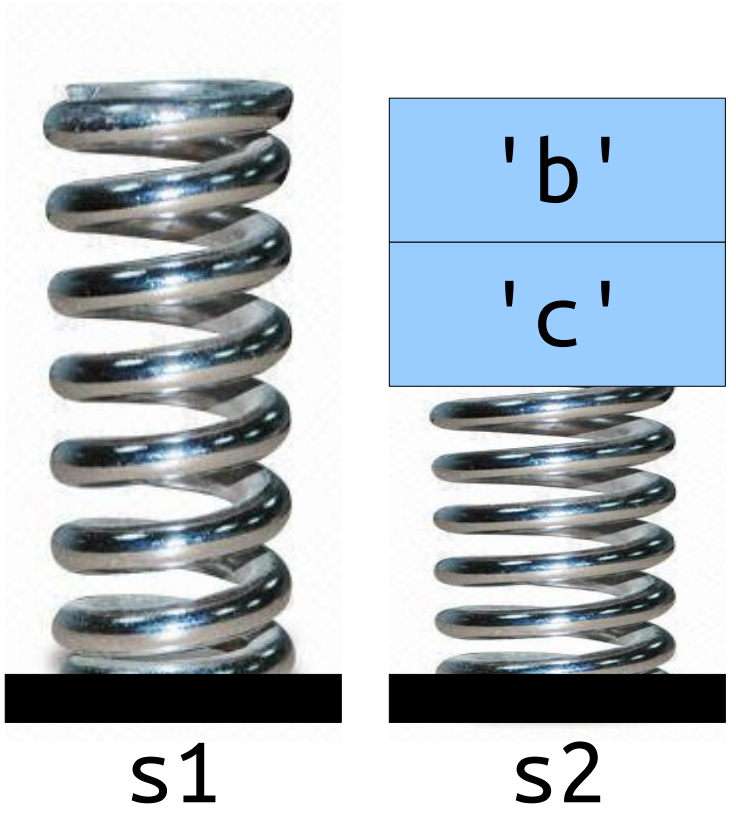


Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

'a'

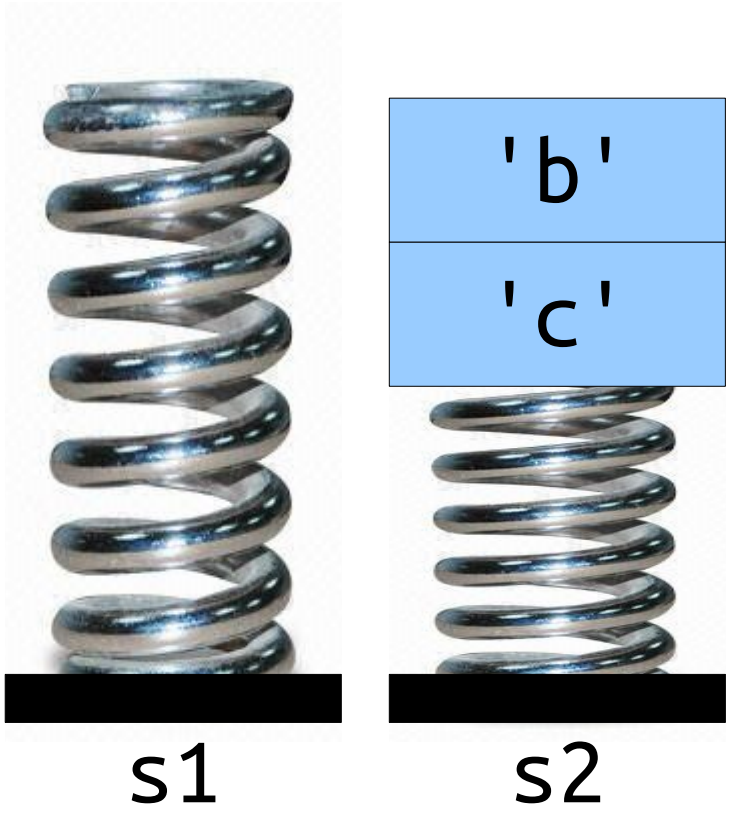


Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

'a'

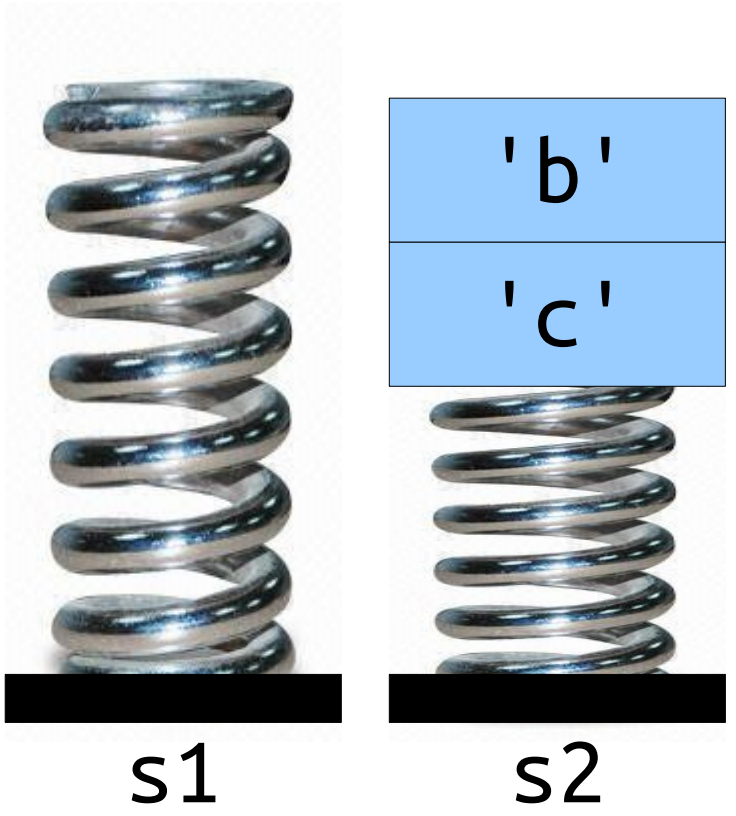


Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

'a'

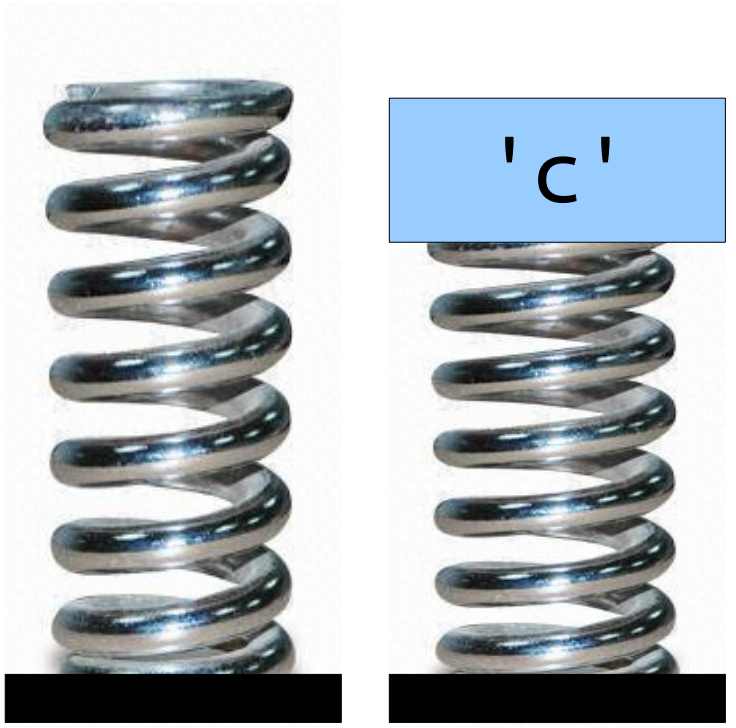


Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

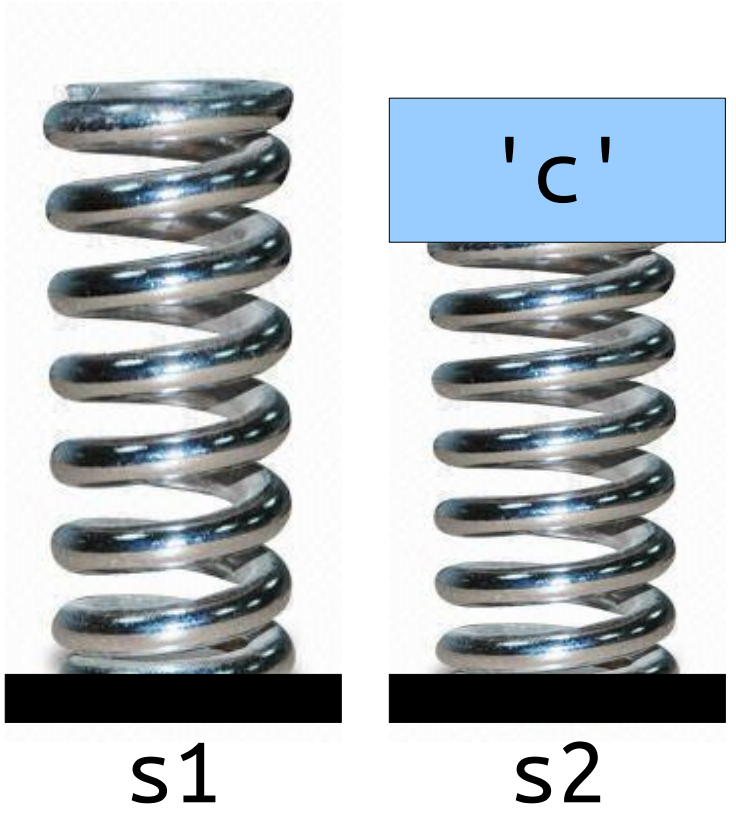
'a'



Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

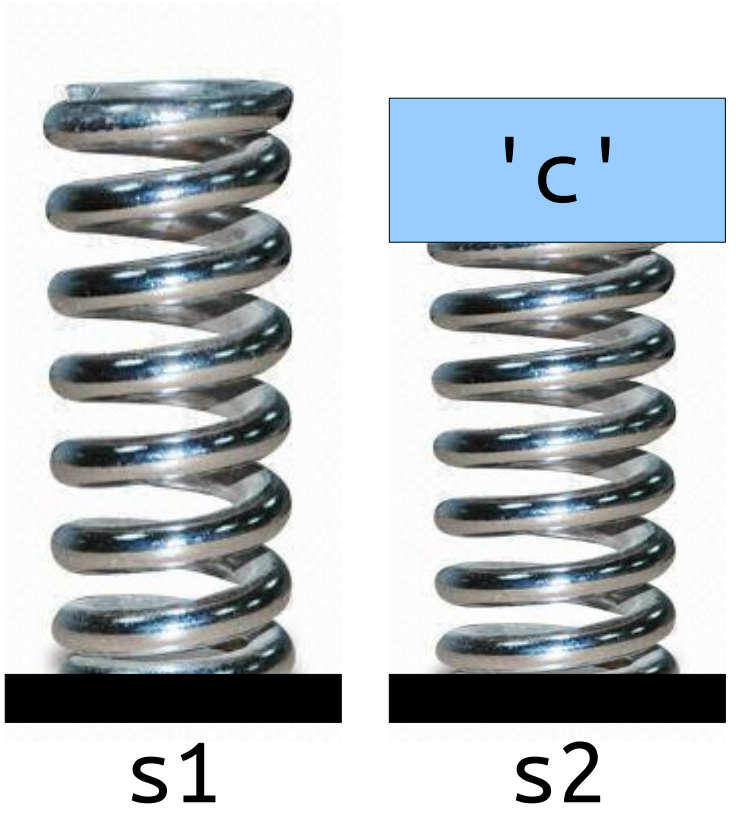


'a'	'b'

Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

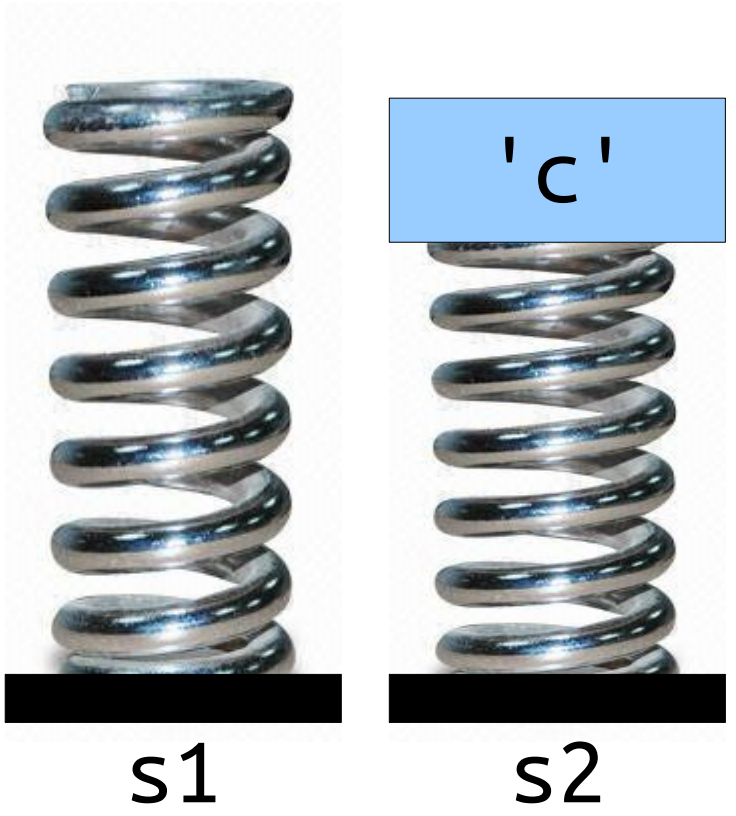


'a'	'b'

Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



'a'	'b'

Stack

What does this code print?

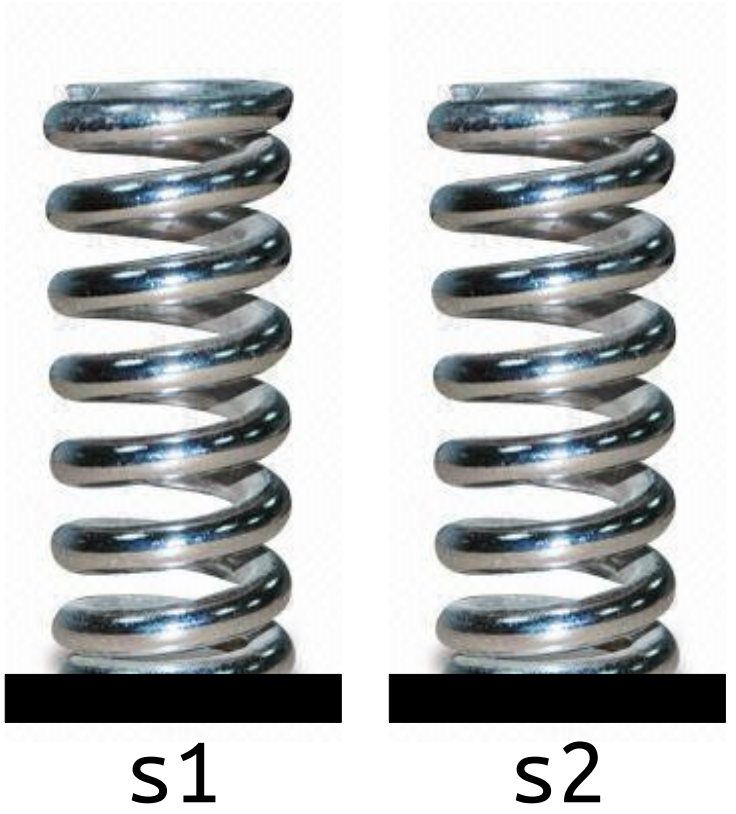
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

'a'	'b'

Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

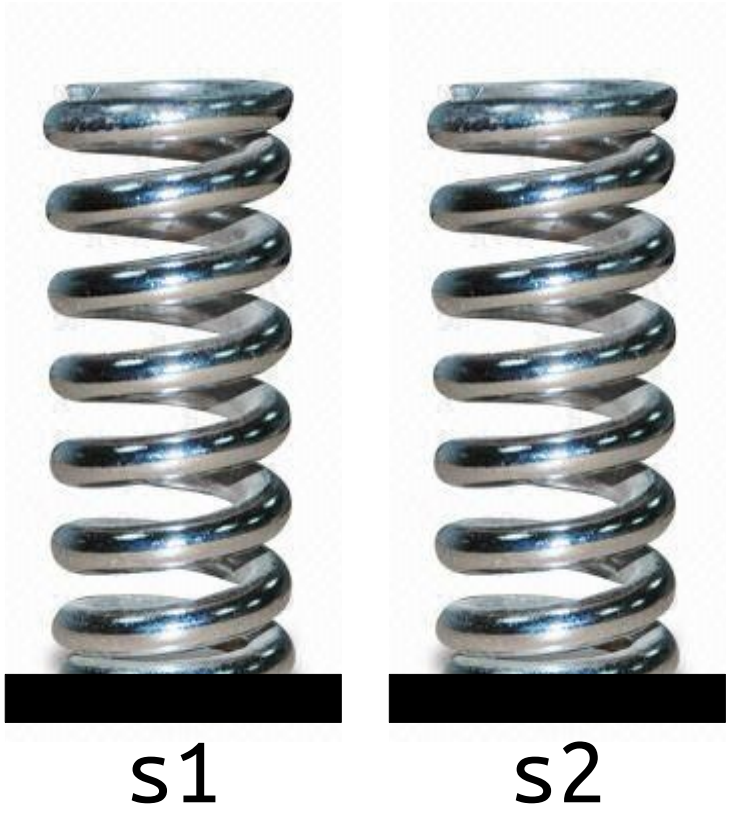


'a'	'b'	'c'

Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```

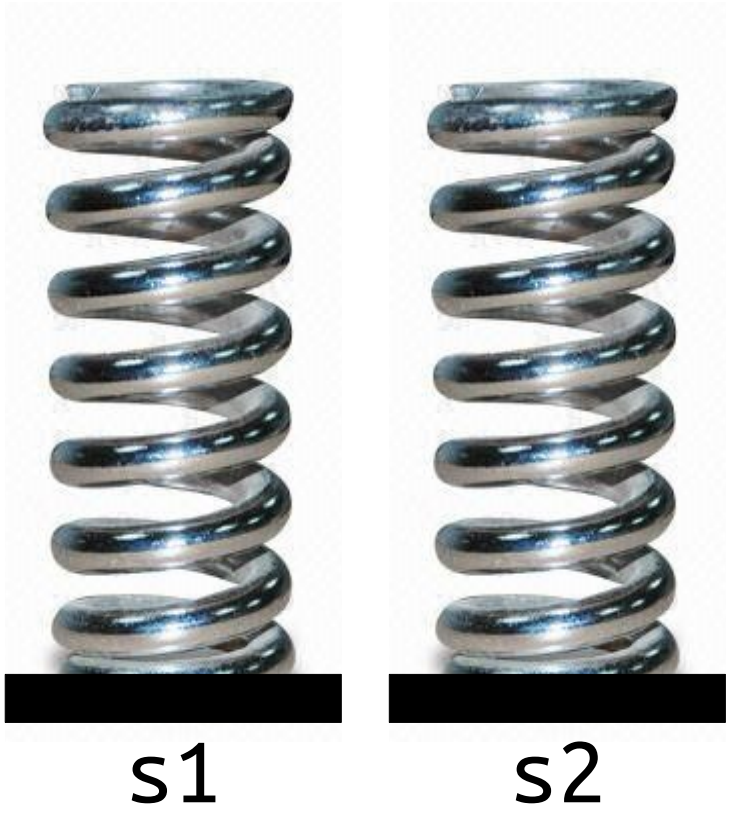


'a'	'b'	'c'

Stack

What does this code print?

```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

while (!s1.isEmpty()) {
  s2.push(s1.pop());
}

while (!s2.isEmpty()) {
  cout << s2.pop() << endl;
}
```



'a'	'b'	'c'

Stack

- Technically speaking, anything you can do with a `Stack` you can also do with a `Vector`.
- So why do we have the `Stack` type as well?
  - ***Clarity:*** Many problems can be modeled elegantly using a stack. Representing those stacks in code with a `Stack` makes the code easier to understand.
  - ***Error-Prevention:*** The `Stack` has fewer operations than a `Vector`. If you're trying to model a stack, this automatically eliminates a large class of errors.
  - ***Efficiency:*** `Stacks` can be slightly faster than `Vectors` because they don't need to support as many operations. (More on that later in the quarter.)

An Application: ***Balanced Parentheses***

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

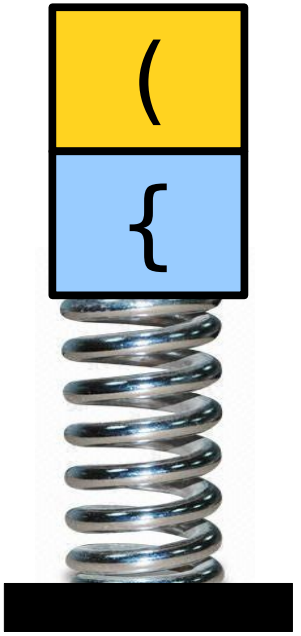
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
            ^
```

Balancing Parentheses

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

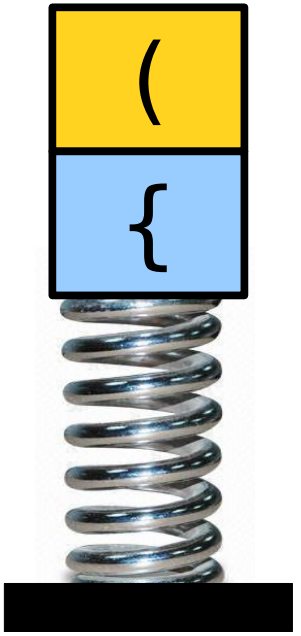
Balancing Parentheses

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



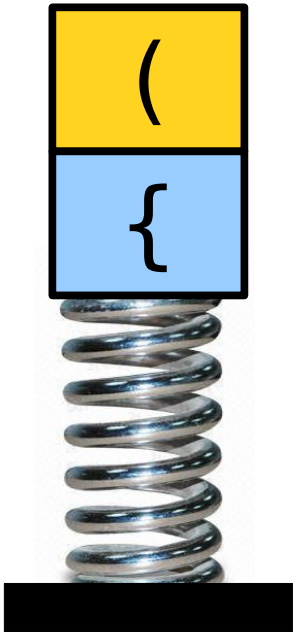
Balancing Parentheses

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



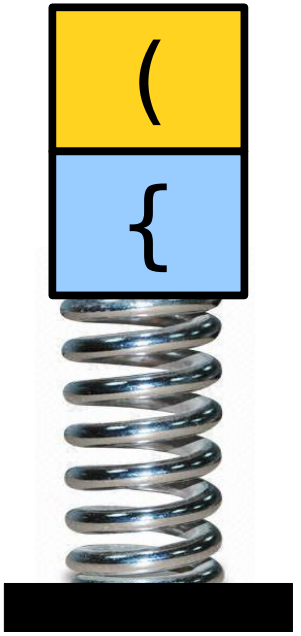
Balancing Parentheses

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



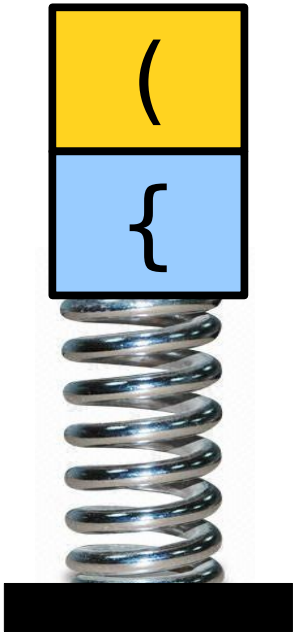
Balancing Parentheses

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



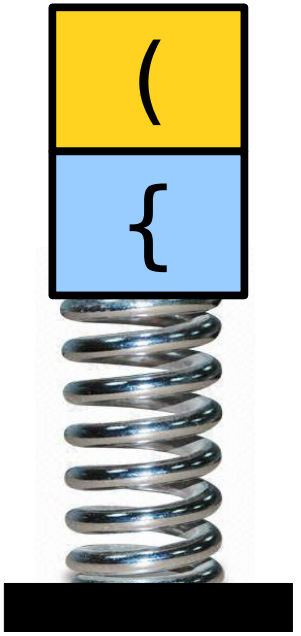
Balancing Parentheses

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



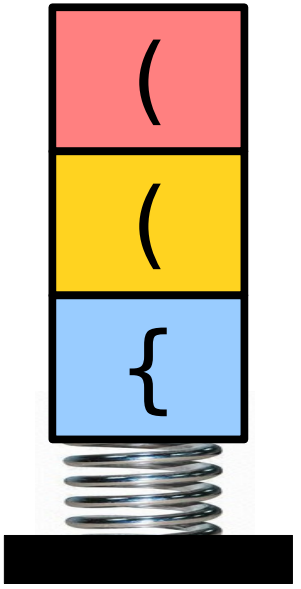
Balancing Parentheses

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



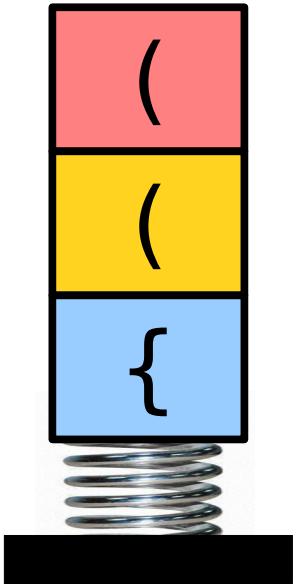
Balancing Parentheses

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



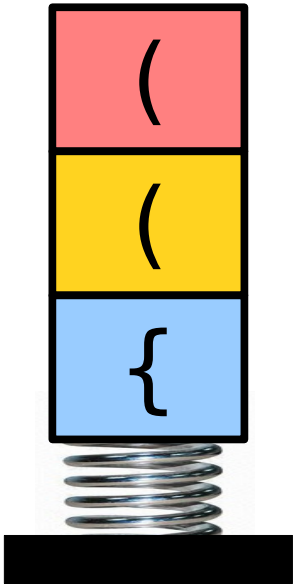
Balancing Parentheses

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



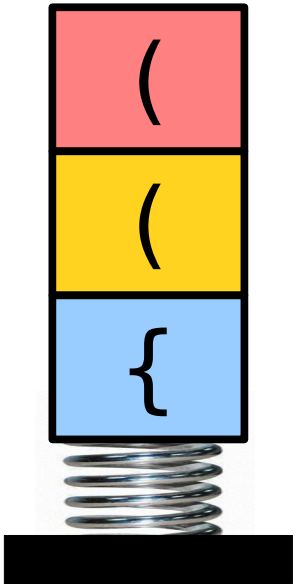
Balancing Parentheses

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



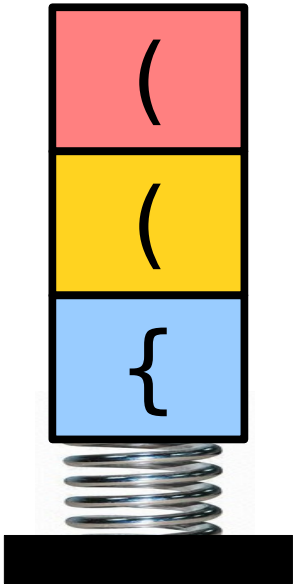
Balancing Parentheses

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



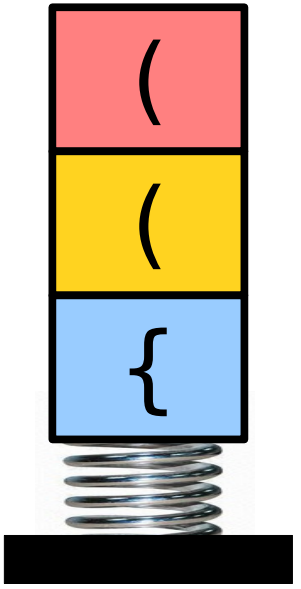
Balancing Parentheses

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



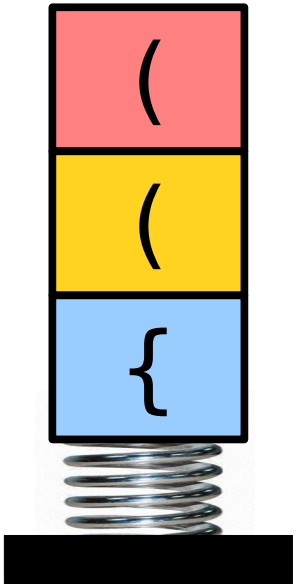
Balancing Parentheses

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



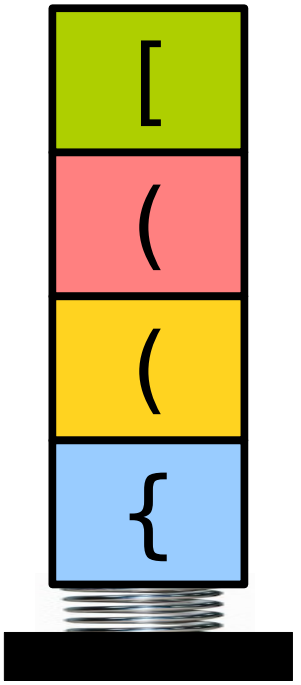
Balancing Parentheses

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



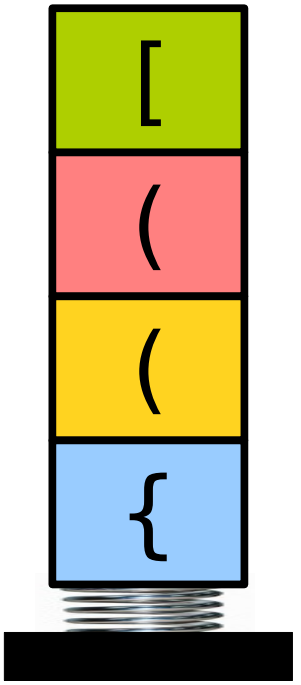
Balancing Parentheses

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



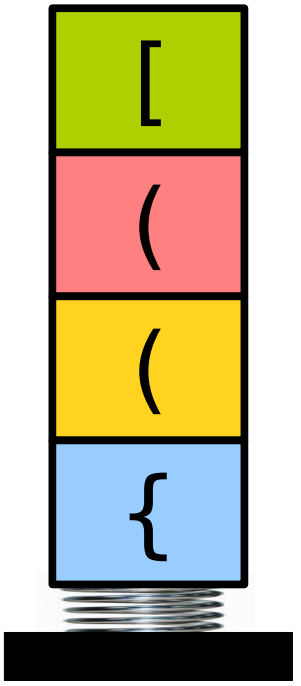
Balancing Parentheses

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



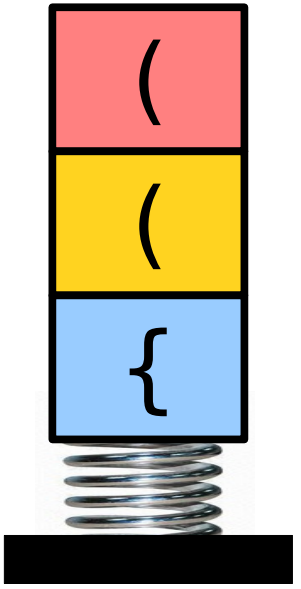
Balancing Parentheses

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



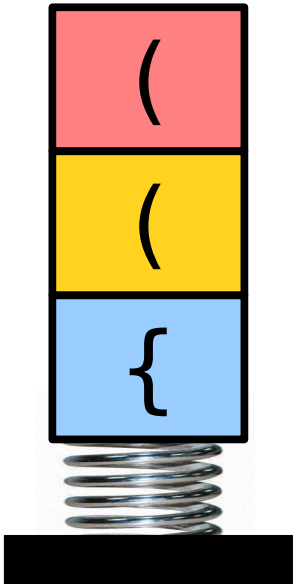
Balancing Parentheses

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



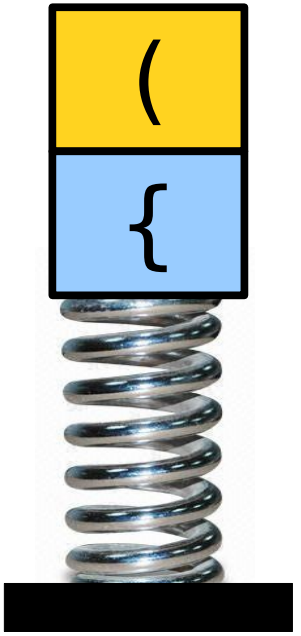
Balancing Parentheses

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



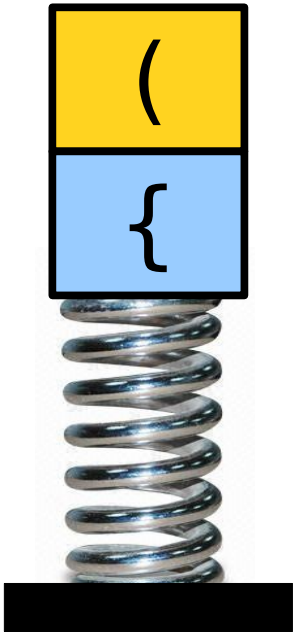
Balancing Parentheses

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



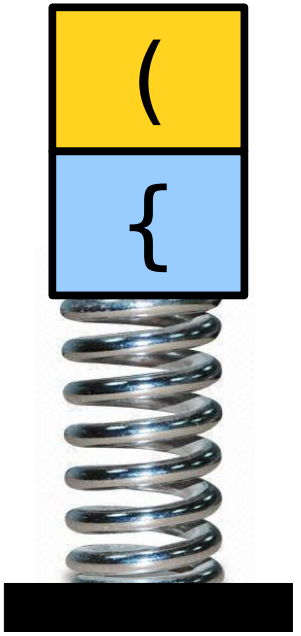
Balancing Parentheses

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



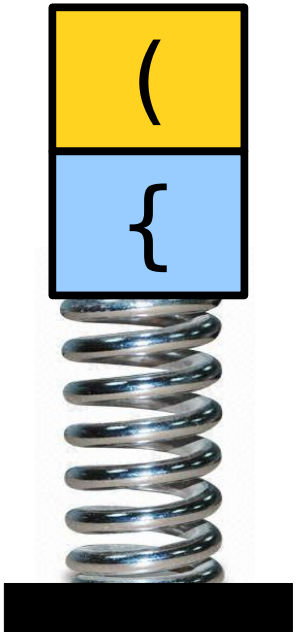
Balancing Parentheses

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



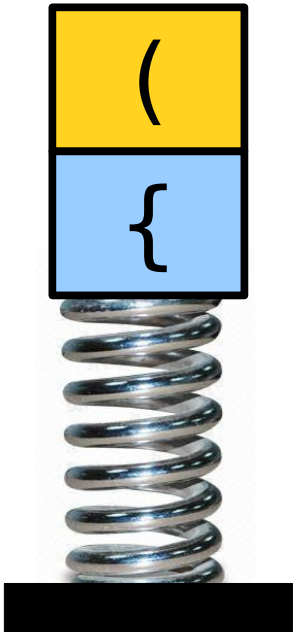
Balancing Parentheses

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



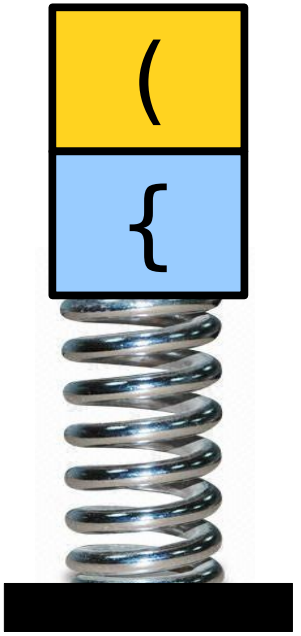
Balancing Parentheses

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



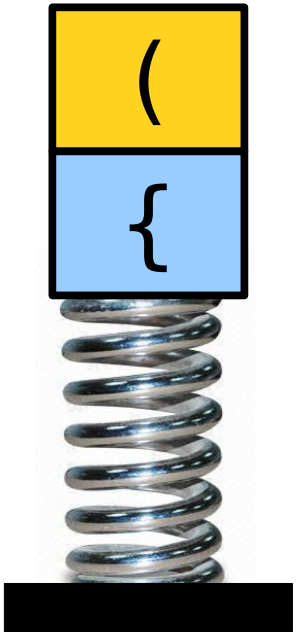
Balancing Parentheses

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



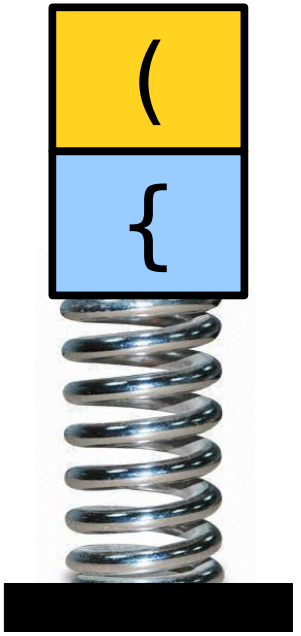
Balancing Parentheses

```
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

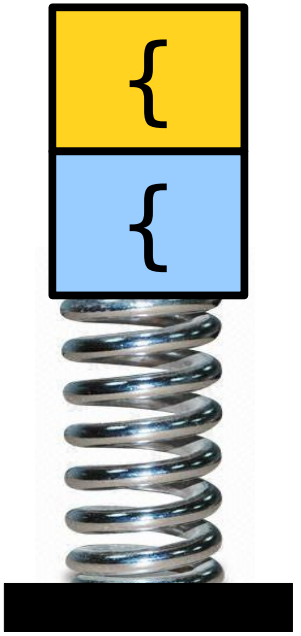
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```

Balancing Parentheses

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

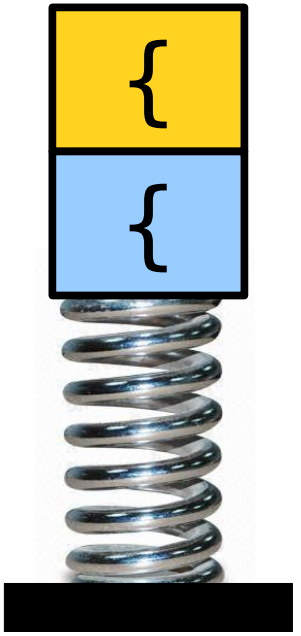
Balancing Parentheses

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



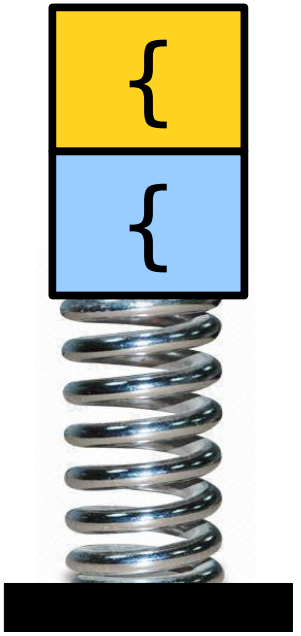
Balancing Parentheses

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



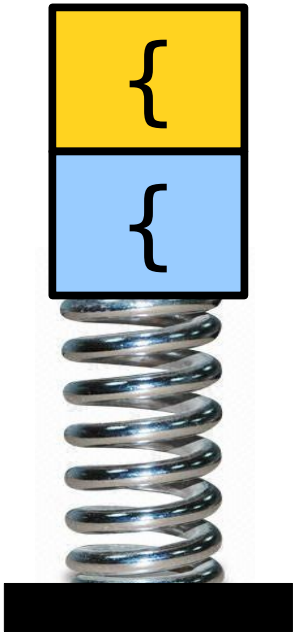
Balancing Parentheses

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



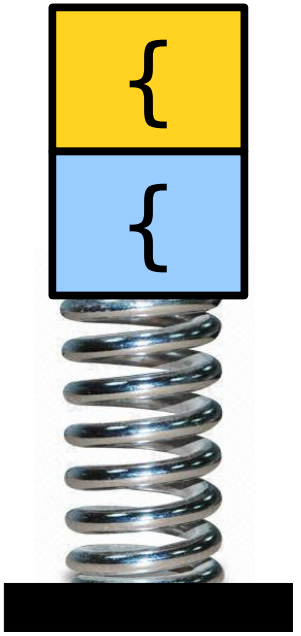
Balancing Parentheses

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



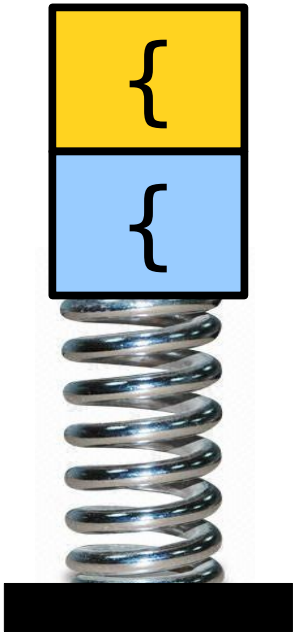
Balancing Parentheses

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



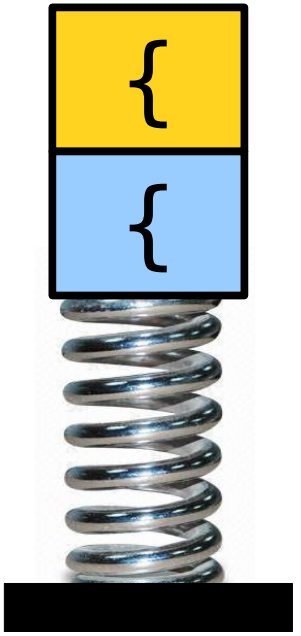
Balancing Parentheses

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



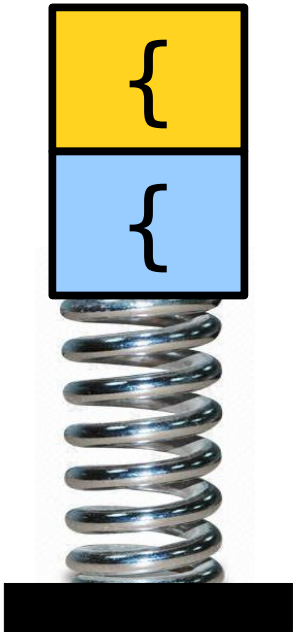
Balancing Parentheses

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



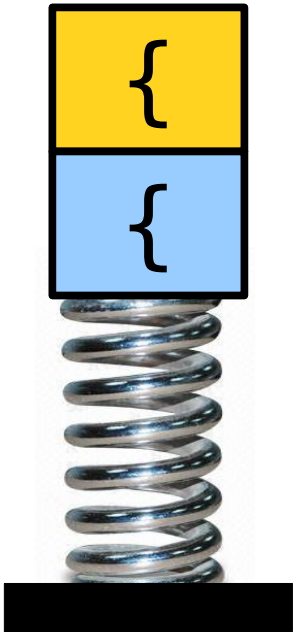
Balancing Parentheses

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



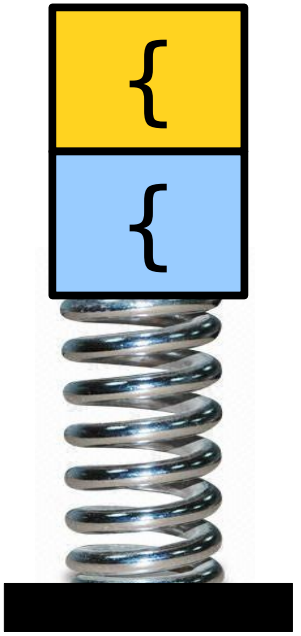
Balancing Parentheses

```
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

([ )]

Balancing Parentheses

```
( [ ) ]
^
```

Balancing Parentheses

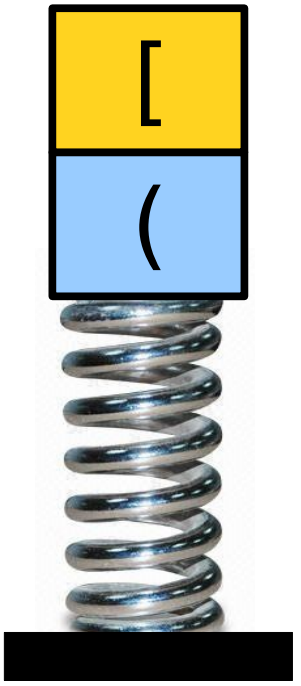
```
( [ ) ]
^
```

Balancing Parentheses

```
( [ ) ]
  ^
```

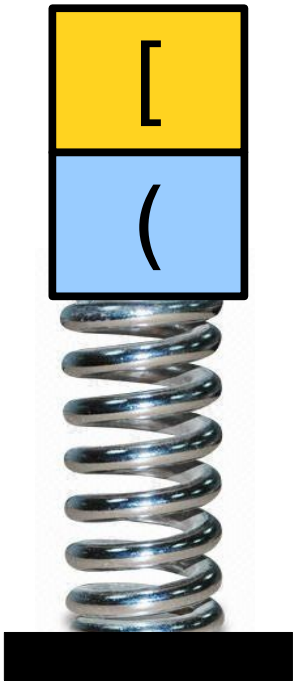
Balancing Parentheses

```
( [ ) ]
  ^
```

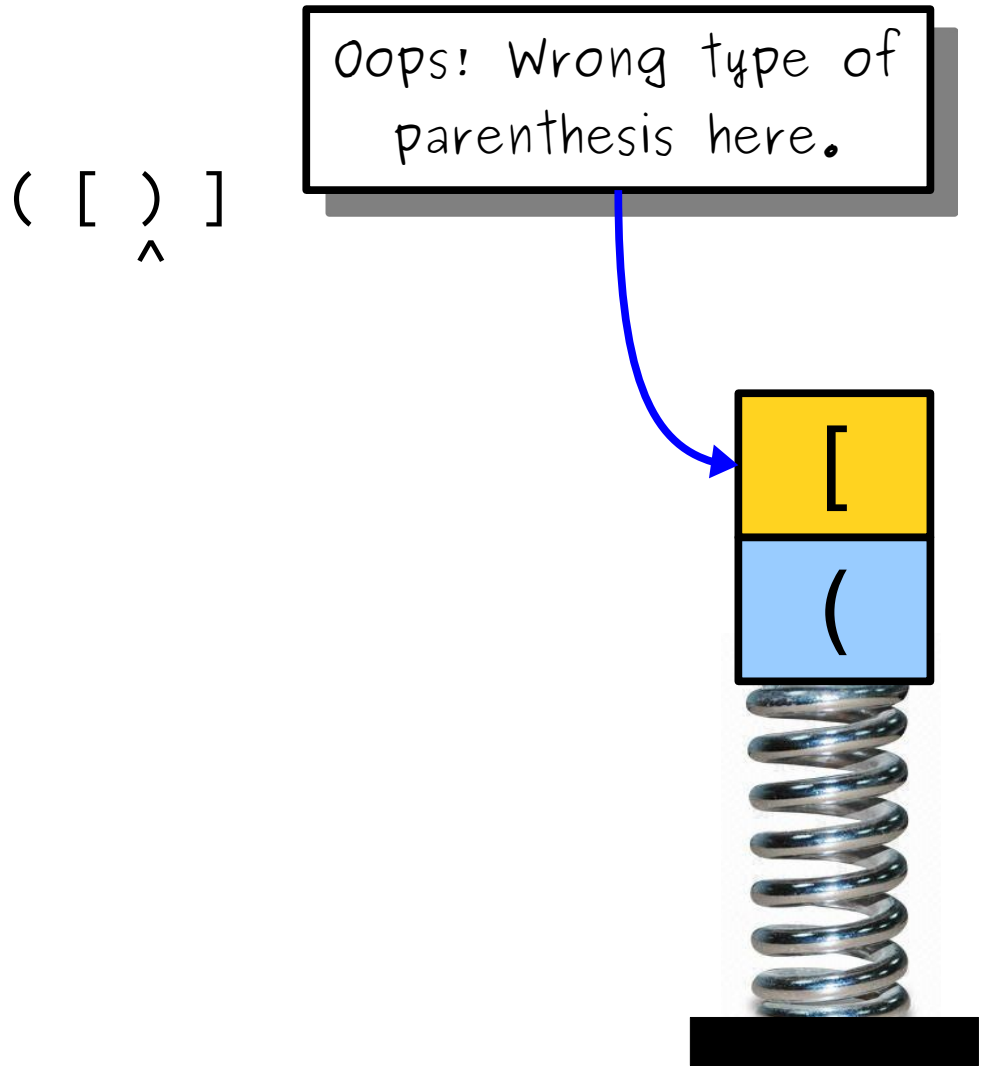


Balancing Parentheses

([ )]
    ^



Balancing Parentheses



Balancing Parentheses

((

Balancing Parentheses

((
^

Balancing Parentheses

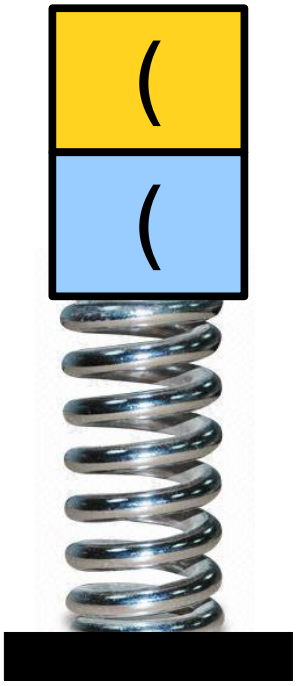
((
^

Balancing Parentheses

((
  ^

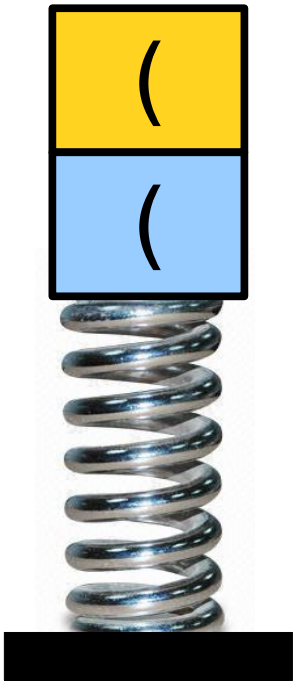
Balancing Parentheses

((
  ^



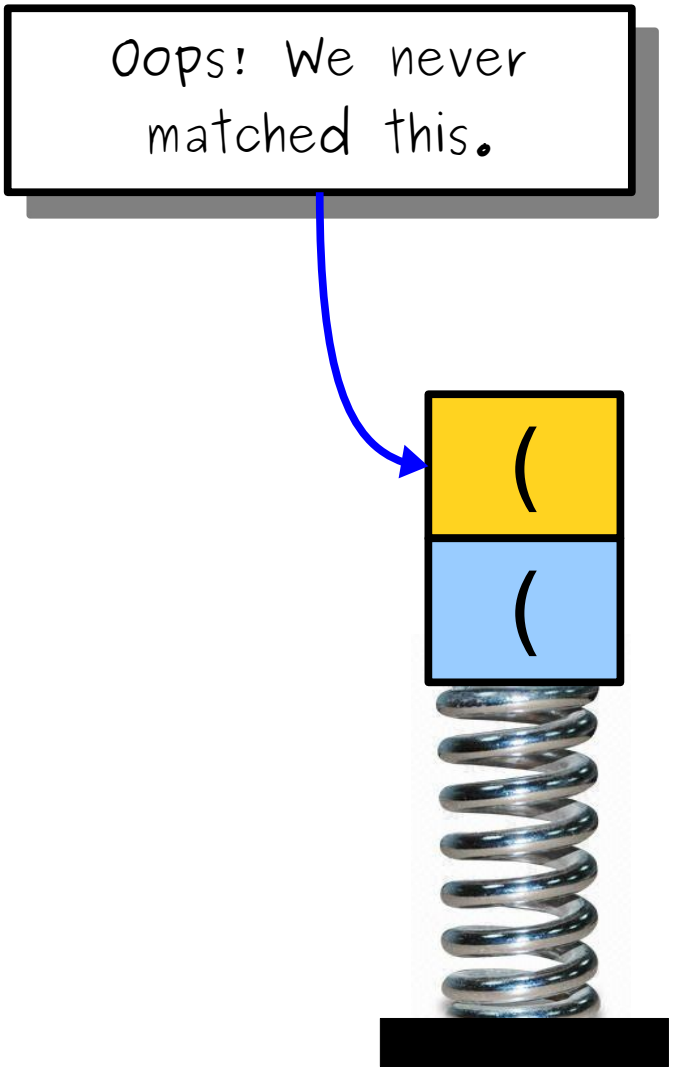
Balancing Parentheses

((



Balancing Parentheses

((



Balancing Parentheses

)

Balancing Parentheses

)
^

Balancing Parentheses

Oops! There's nothing on the stack to match.

)
^

Our Algorithm

- For each character:
  - If it’s an open parenthesis or brace, push it onto the stack.
  - If it’s a close parenthesis or brace:
    - If the stack is empty, report an error.
    - If the character doesn’t pair with the character on top of the stack, report an error.
- At the end, return whether the stack is empty (nothing was left unmatched).

More `Stack` Applications

- Stacks show up all the time in ***parsing***, recovering the structure in a piece of text.
  - Often used in natural language processing; take CS224N for details!
  - Used all the time in compilers – take CS143 for details!
  - There's a deep theorem that says that many structures appearing in natural language are perfectly modeled by operations on stacks; come talk to me after class if you're curious!
- They're also used as building blocks in larger algorithms for doing things like
  - making sure a city's road networks are navigable (finding *strongly connected components;* take CS161 for details!) and
  - searching for the best solution to a problem – stay tuned!

Time-Out for Announcements!

MLK Weekend

- Some suggested reading / listening / watching recommendations:
  - “The Autobiography of Malcolm X,” as told to Alex Haley.
  - “The Ballot or the Bullet” by Malcolm X.
  - “Between the World and Me” by Ta-Nehisi Coates.
  - “The Case for Reparations” by Ta-Nehisi Coates.
  - “Debate at Cambridge Union,” James Baldwin and William F. Buckley, Jr.
  - “Do Artifacts Have Politics?” by Langdon Winner.
  - “Letter from Birmingham City Jail” by Martin Luther King, Jr.
  - “Letter from a Region in my Mind” by James Baldwin.
  - “Notes on an Imagined Plaque” by The Memory Palace.
  - “The Other America” by Martin Luther King, Jr.

Asynchronous Lecture

- We will not have class this upcoming Monday in observance of the MLK holiday.
- Monday’s lecture will instead be prerecorded and available online on Canvas starting at around 5PM today.
- Watch that lecture before we return for Wednesday’s (in-person) lecture.

Assignment 2

- Assignment 1 was due today a 1:00PM.
  - Need more time? Use one late day to extend the deadline by 24 hours or two to extend it by 48 hours.
- Assignment 2 (***Fun With Collections***) goes out today. It's due next Friday at 1:00PM.
  - Use collections to learn what language a text is written in – and expand your mind about the world of human language!
  - Explore the impact of sea level rise on coastal regions!
- Have questions?
  - Stop by the LaIR! Or ask on EdStem! Or email your section leader!

Assignment 2

- This assignment contains a series of short-answer ethics questions designed to get you thinking about the social impact of computing.
- It’s critical to think about the effect your software has on others, especially given the scale of modern software systems.
- These will form a part of your grade on the assignment separately from your functionality and style scores.

YEAH Hours

- We will be holding YEAH hours on ***Fridays*** from ***4:30PM – 5:30PM***.
- Today’s will be in ***Durand 450***.
- These are purely optional, but are great ways to get an overview of the assignment before you dive into it.
- Sessions will be recorded, and slides will be made available for folks who can’t make it.

Discussion Sections

- Discussion sections have started! You should have received an email with your section time and section leader's name.
- Don't have a section? You can sign up for any open section by visiting

  https://cs198.stanford.edu/

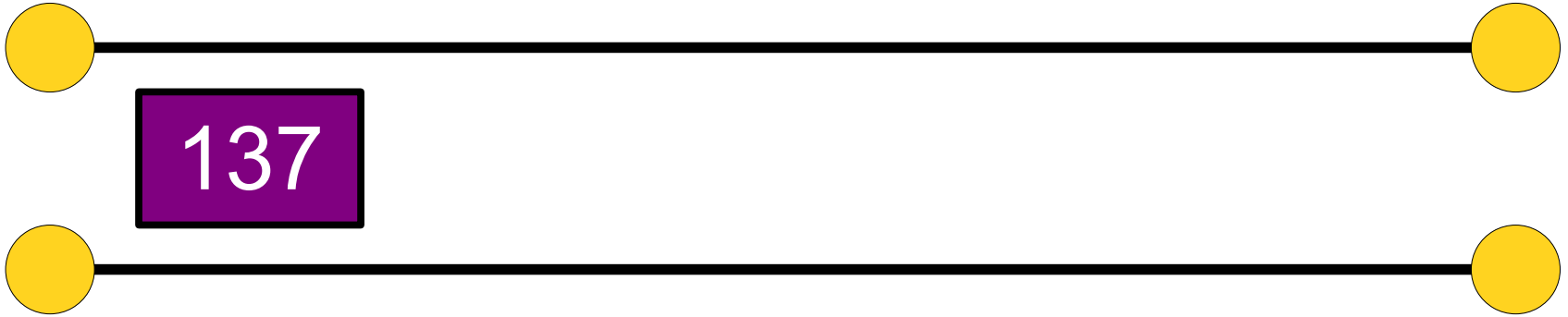
  logging in via "CS106 Sections Login," and picking a section of your choice.
- Attendance is required.
  - If you have a recurring conflict, contact Jonathan to discuss a permanent swap.
  - If you have one-off conflicts, email your section leader at least 24 hours in advance.

```
lecture.pop();
```

Queue

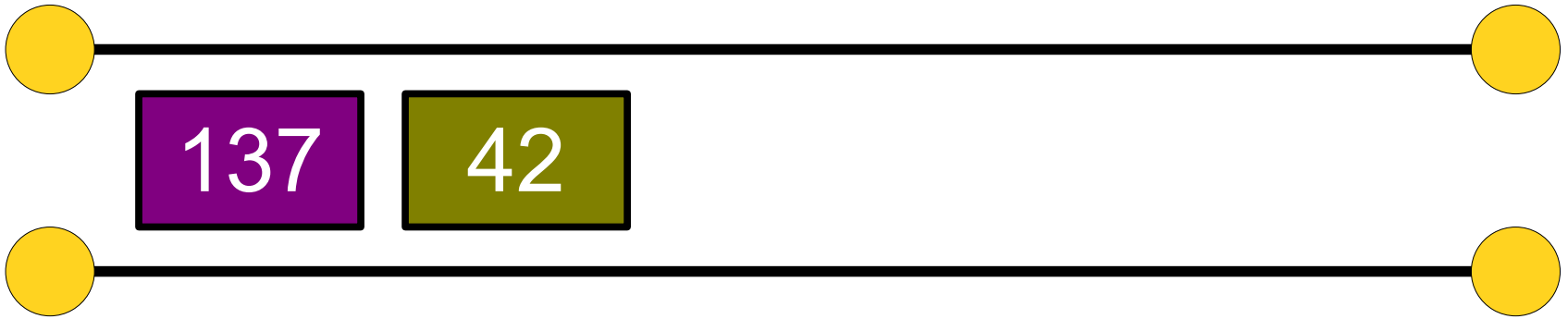
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



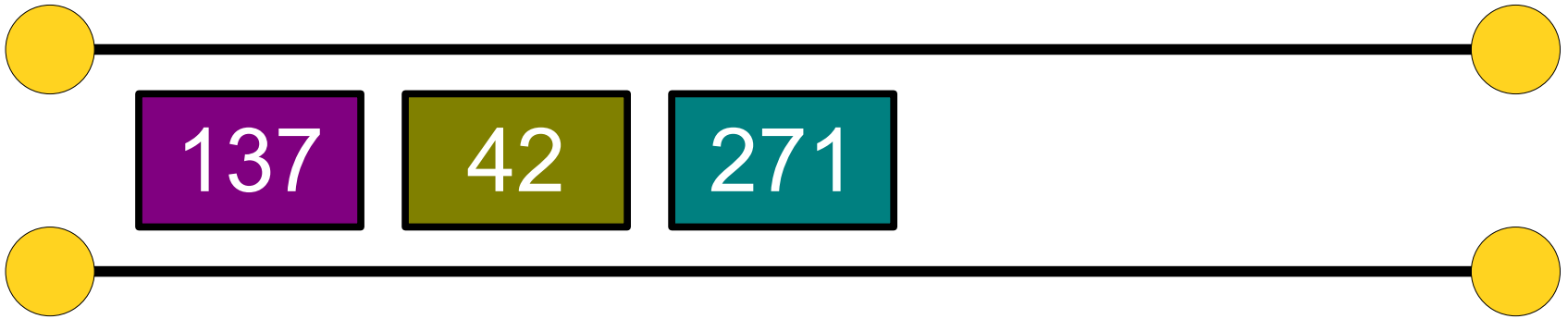
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



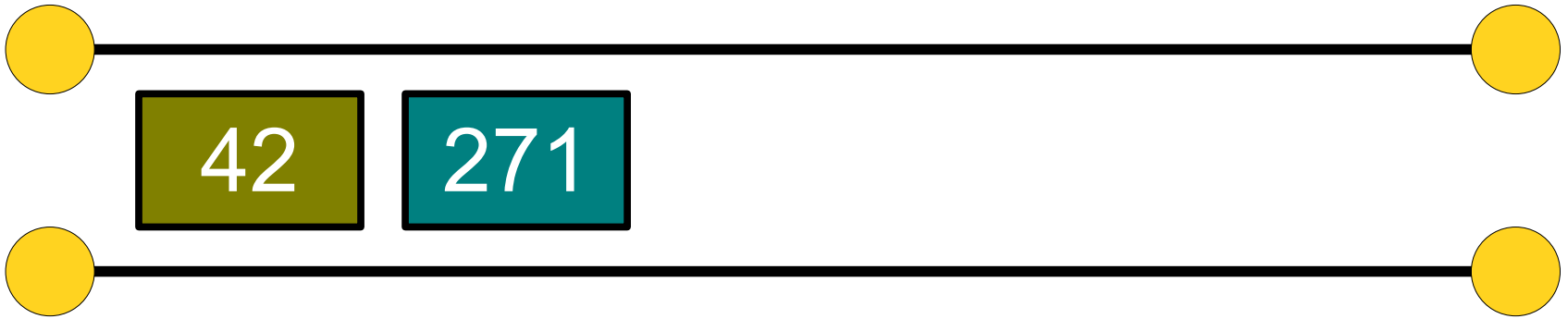
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



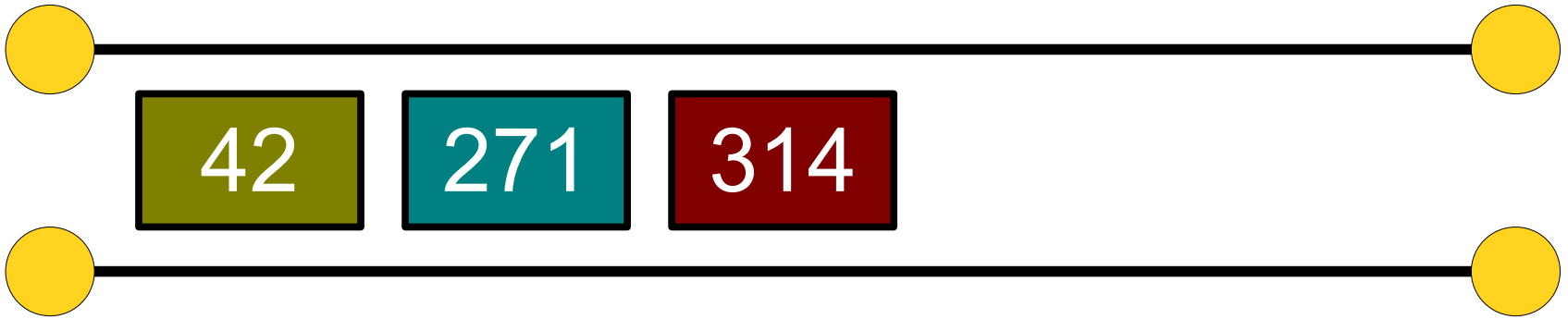
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



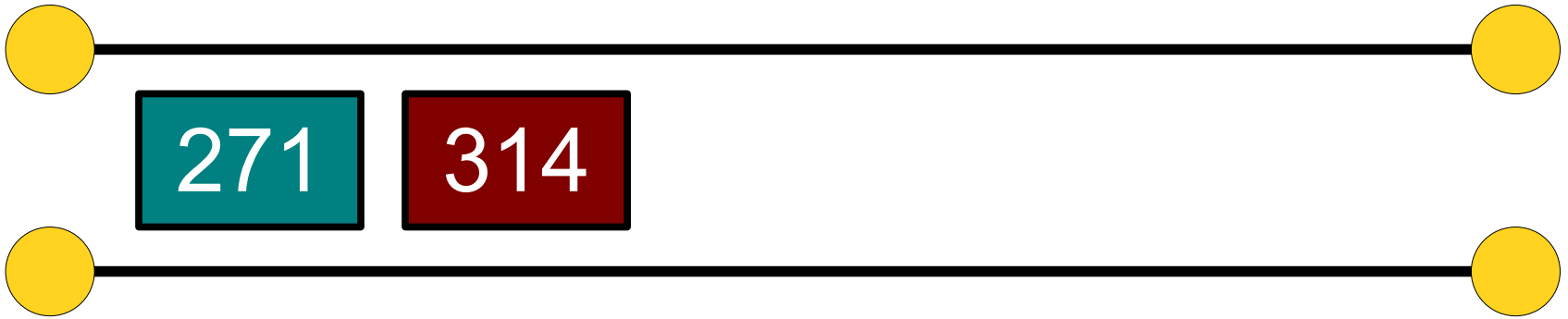
Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.



Queue

- A **`Queue`** is a data structure representing a waiting line.
- Objects can be ***enqueued*** to the back of the line or ***dequeued*** from the front of the line.
- No other objects in the queue are visible.
- Example: A checkout counter.

Queue

- What does this code print?

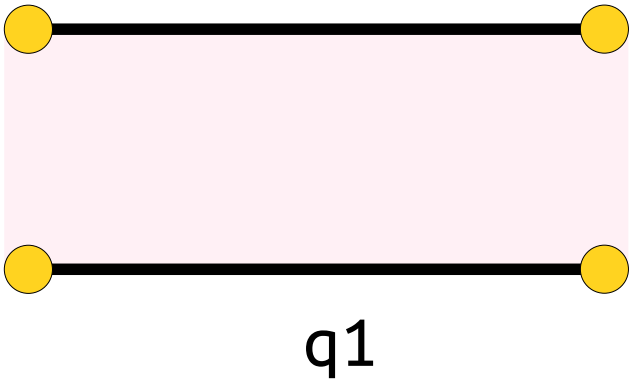
```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

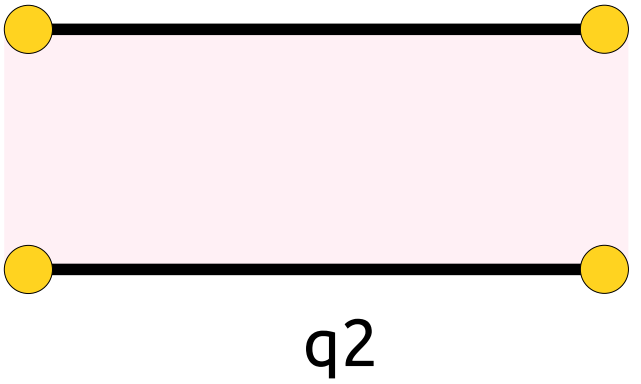
Answer at
https://cs106b.stanford.edu/pollev

Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

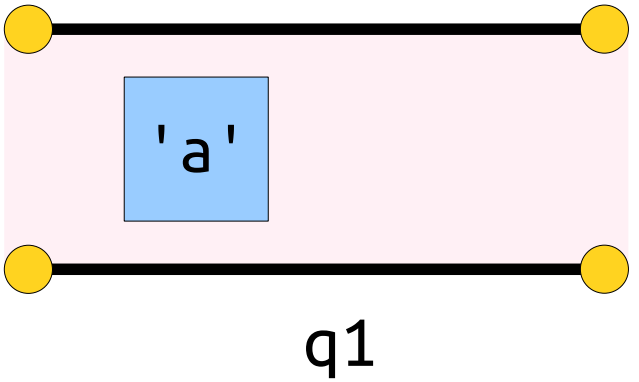


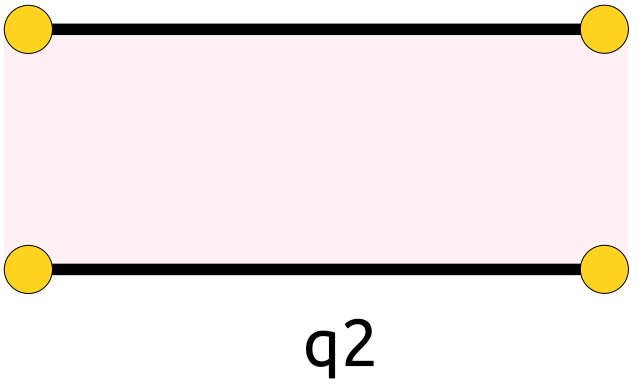


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

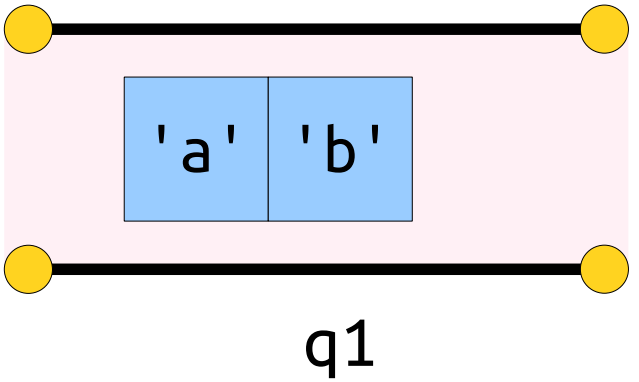


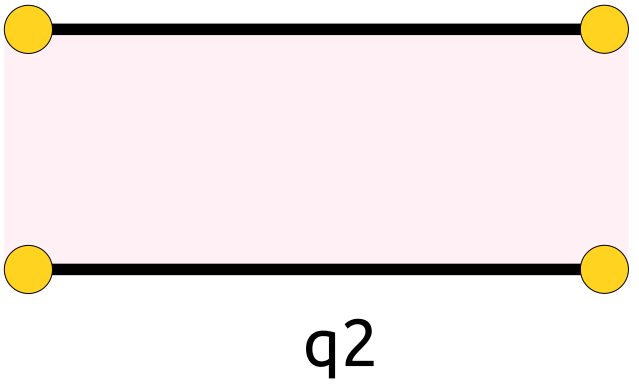


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

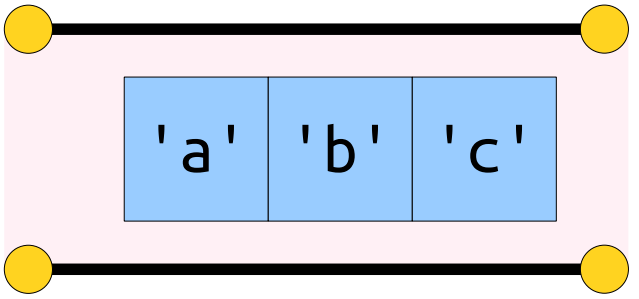




Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



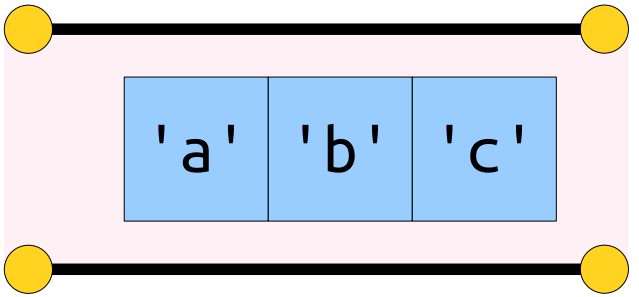
q1

q2

Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



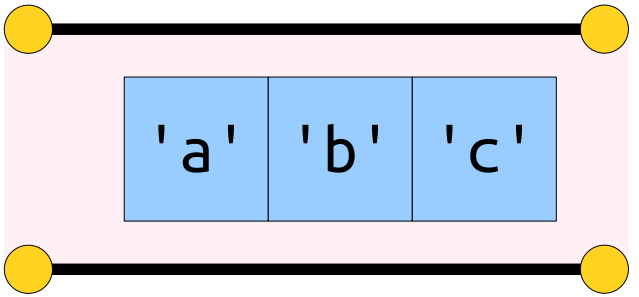
q1

q2

Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



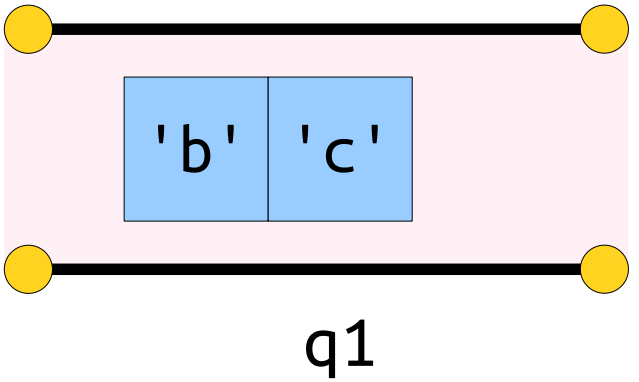
q1

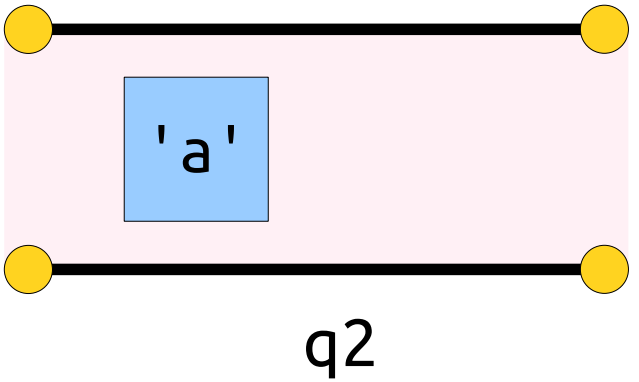
q2

Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

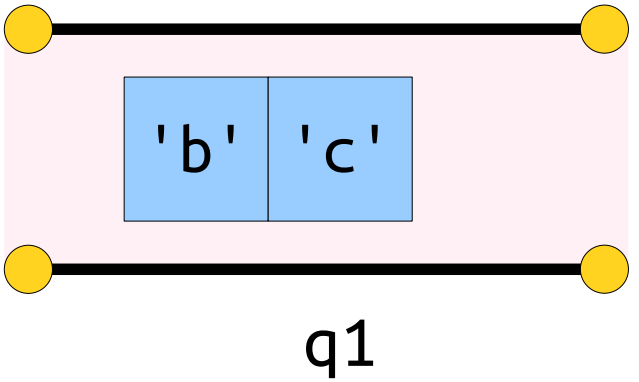


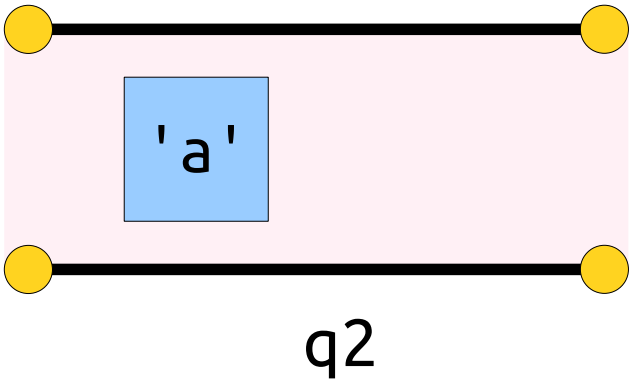


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');
while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}
while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

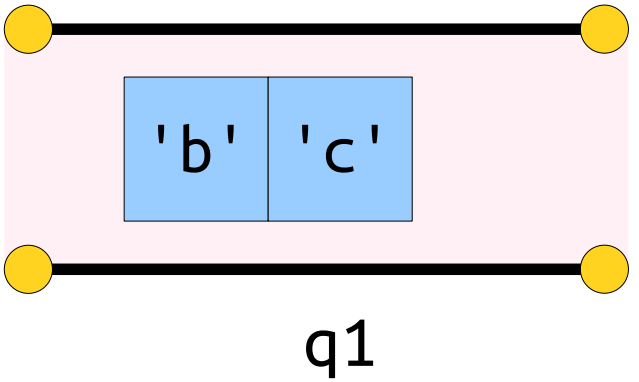


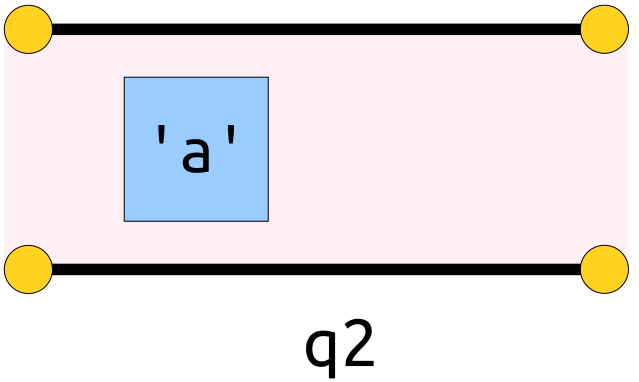


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

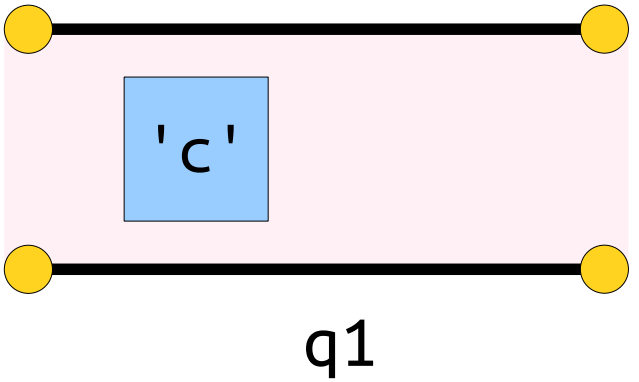


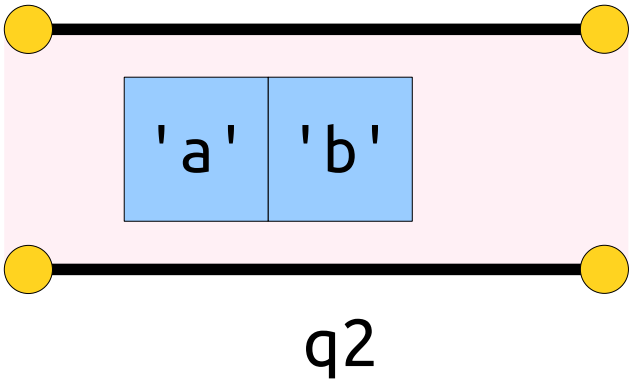


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

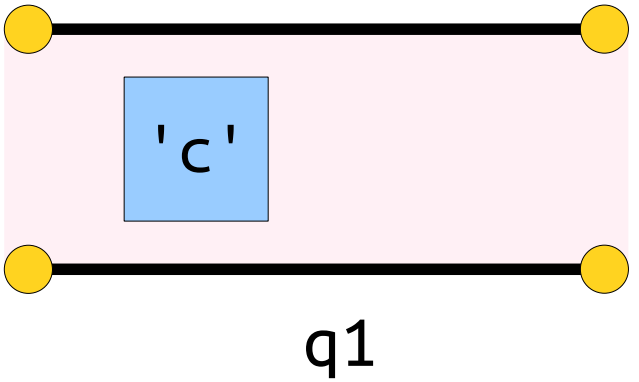


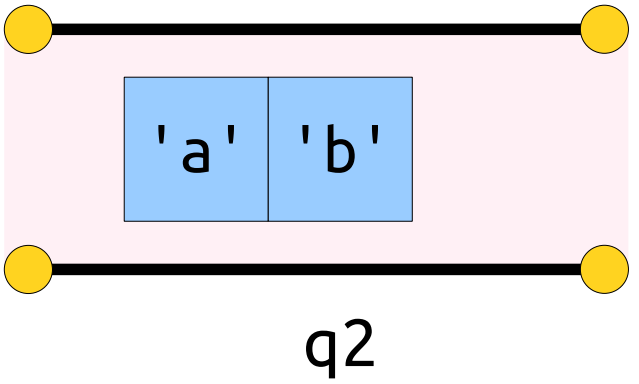


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');
while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}
while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

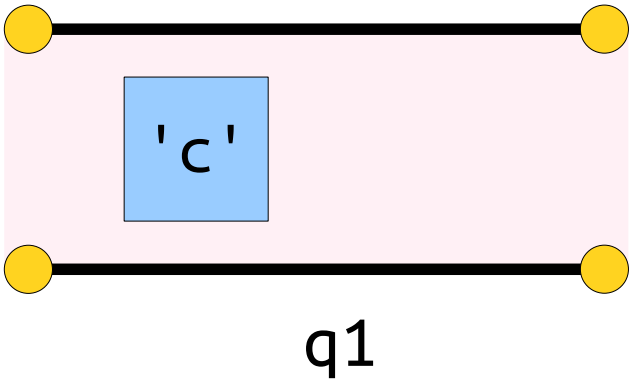


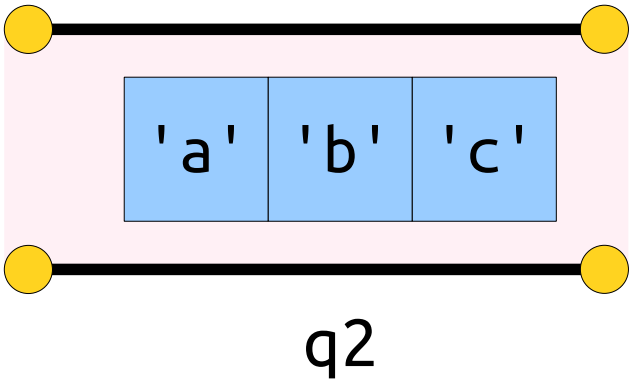


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

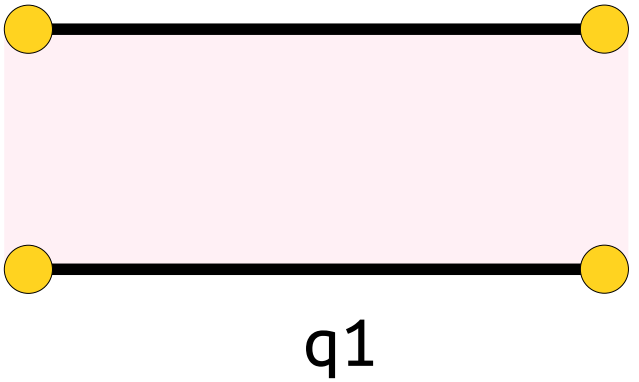


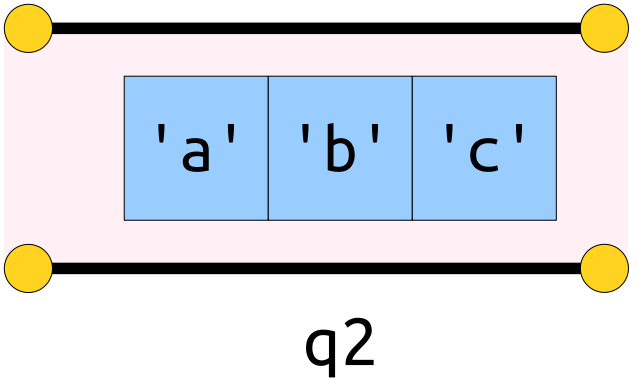


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');
while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}
while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

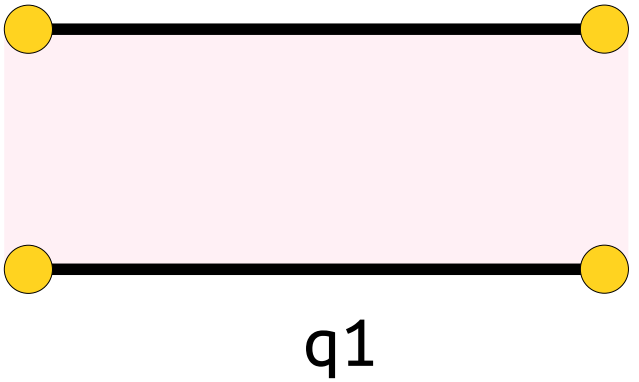




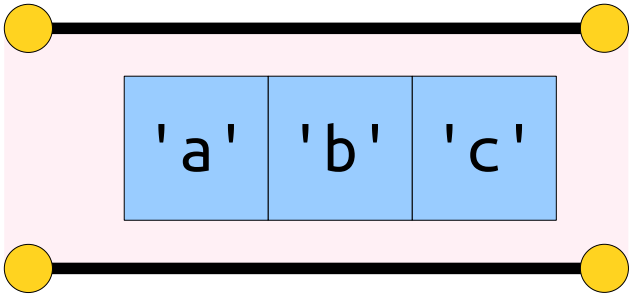
Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



q1

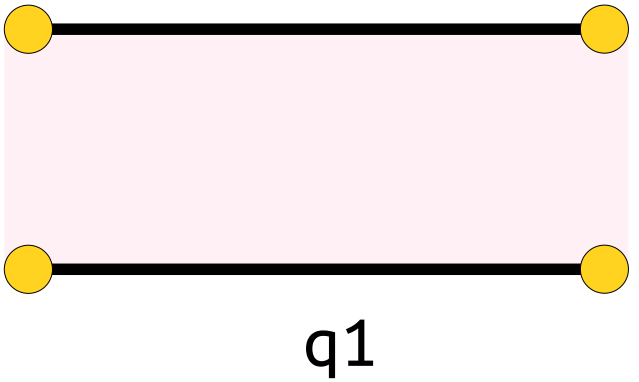


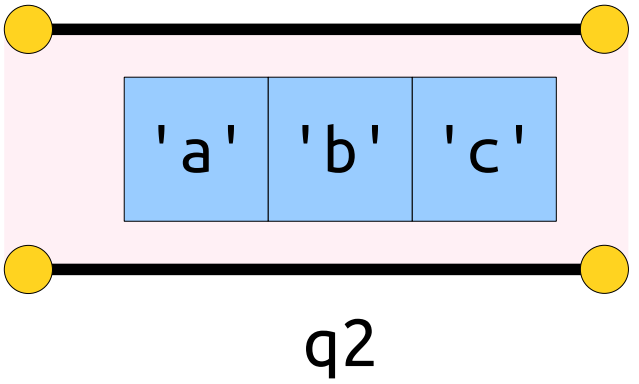
q2

Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



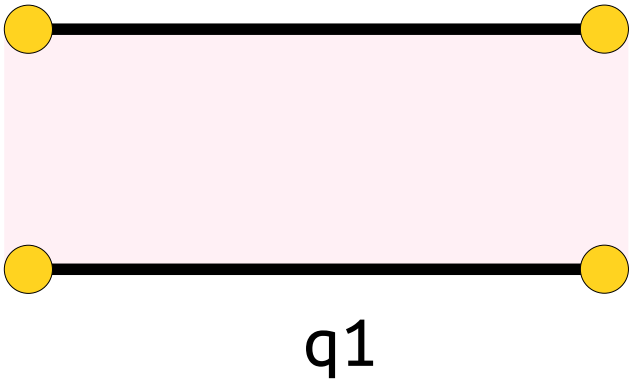


Queue

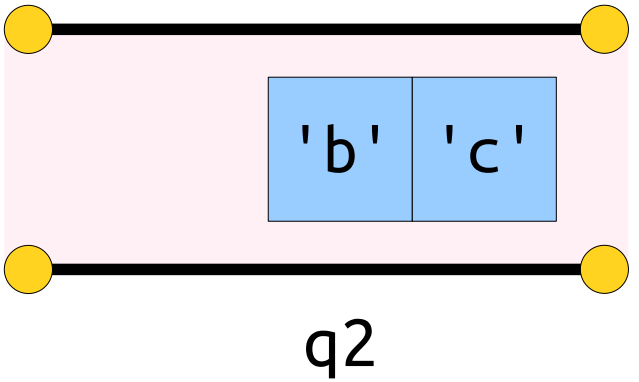
- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

'a'



q1



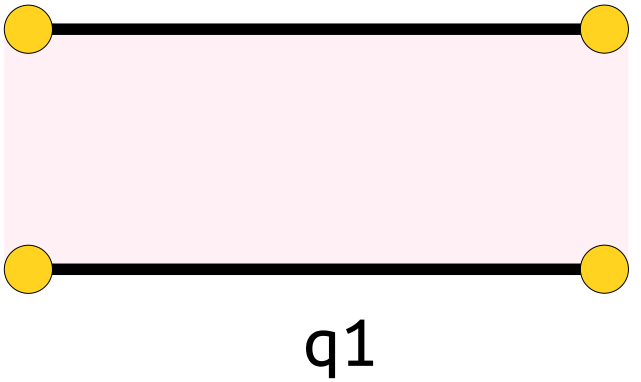
q2

Queue

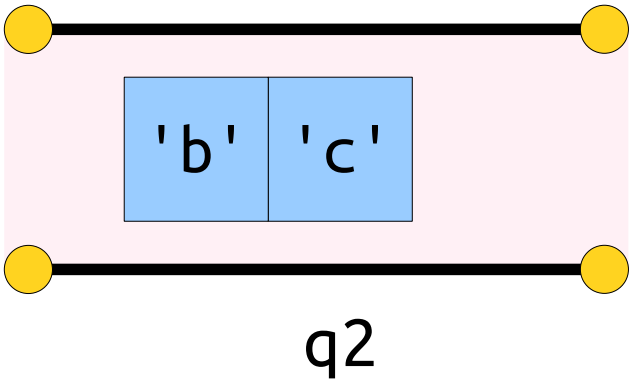
- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

'a'



q1



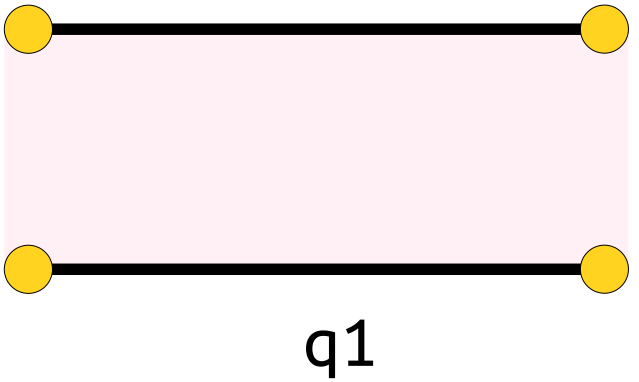
q2

Queue

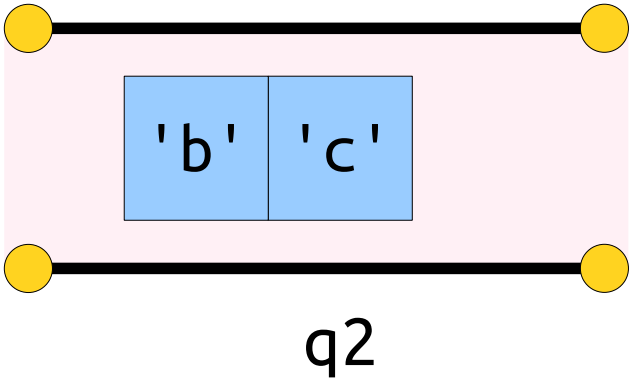
- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

'a'



q1



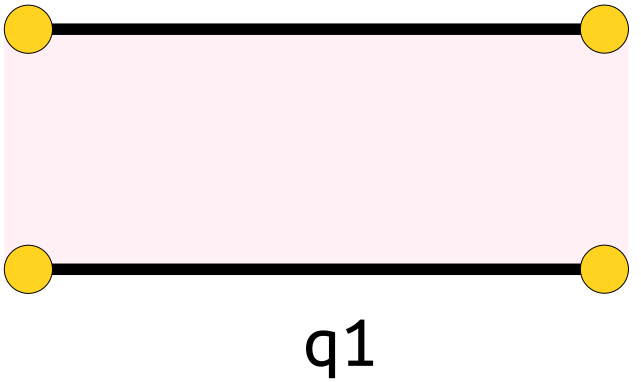
q2

Queue

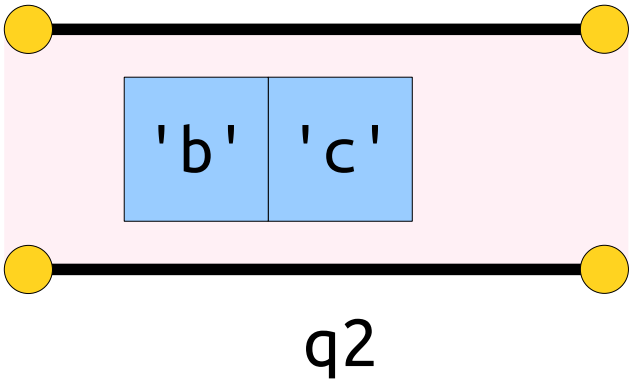
- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```

'a'



q1

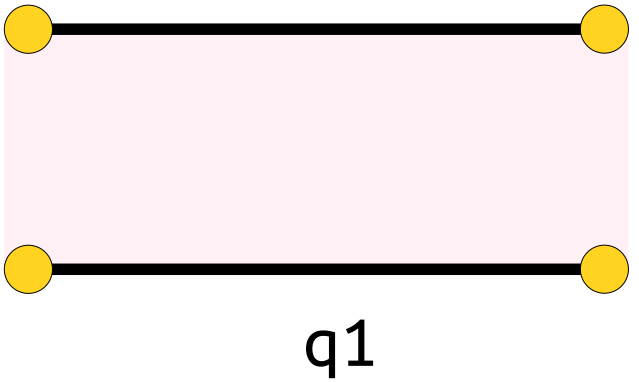


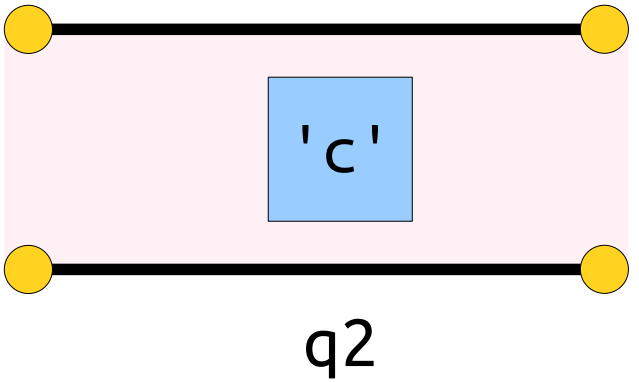
q2

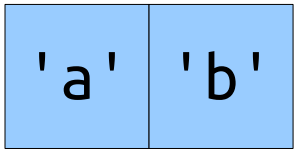
Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



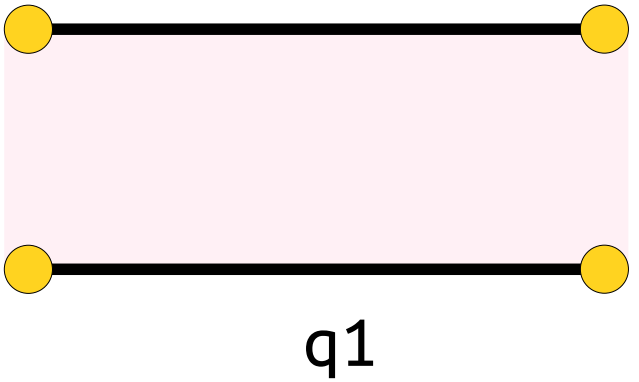


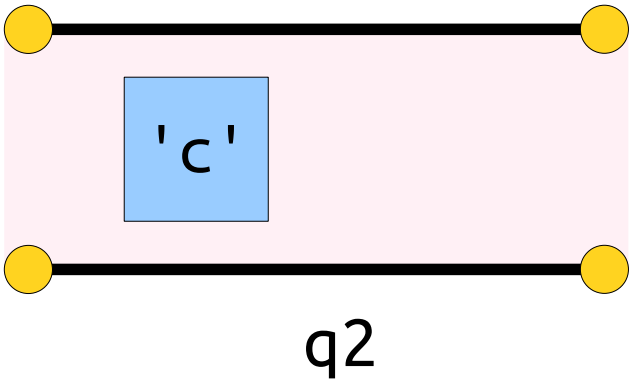


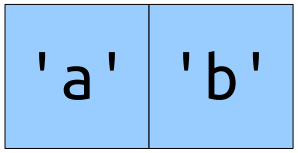
Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



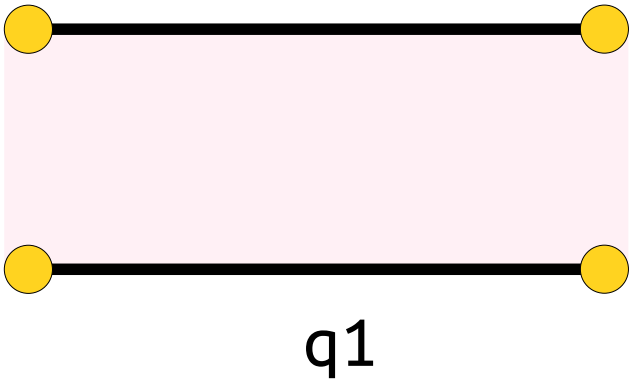


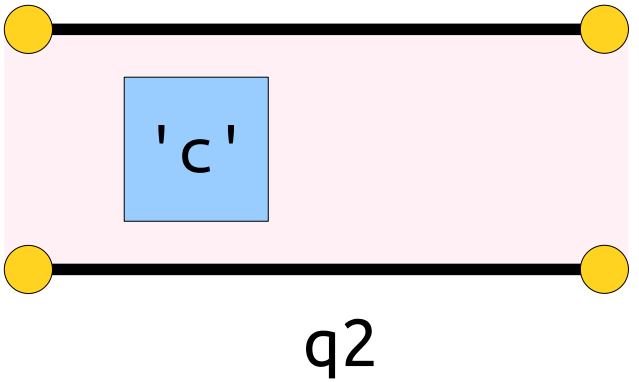


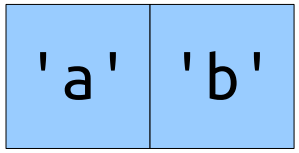
Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



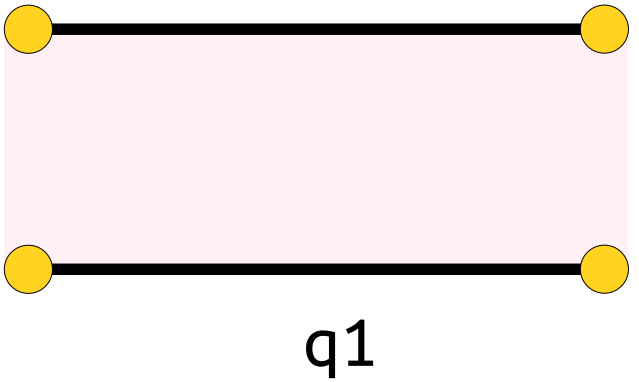


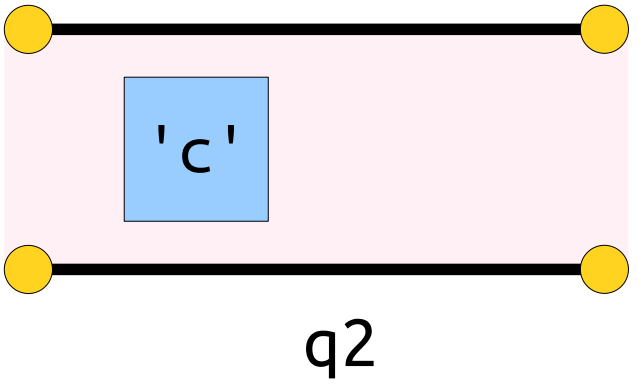


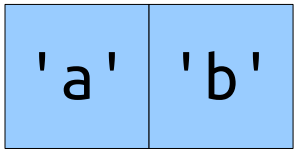
Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



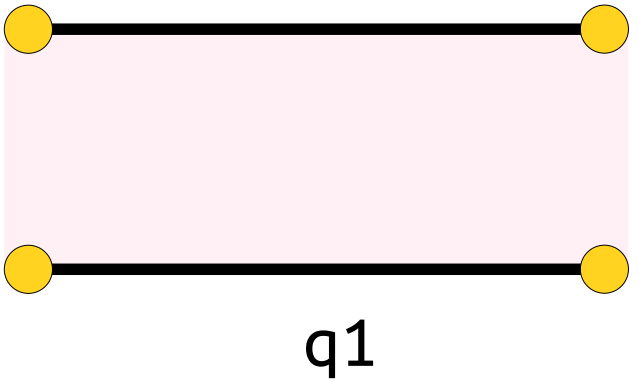


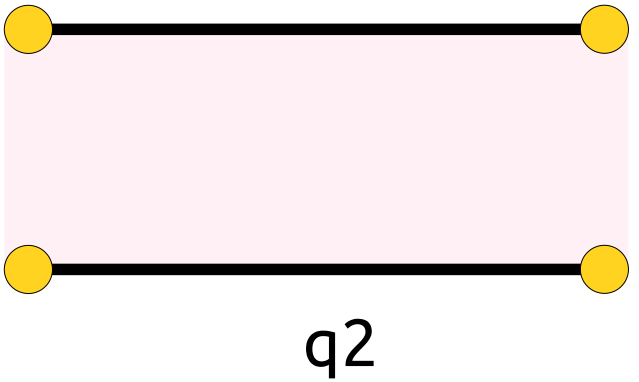


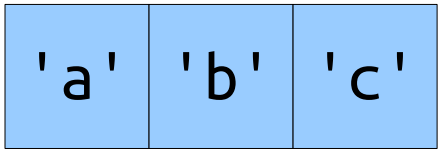
Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



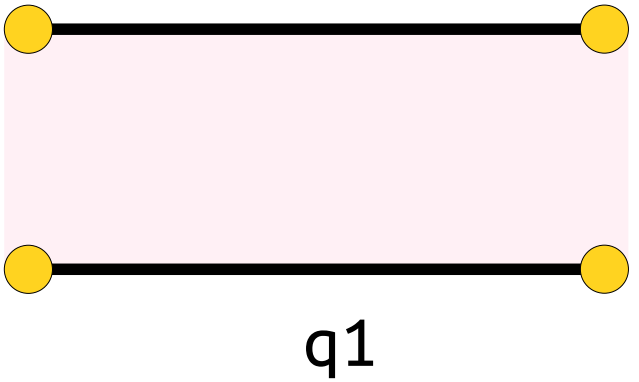


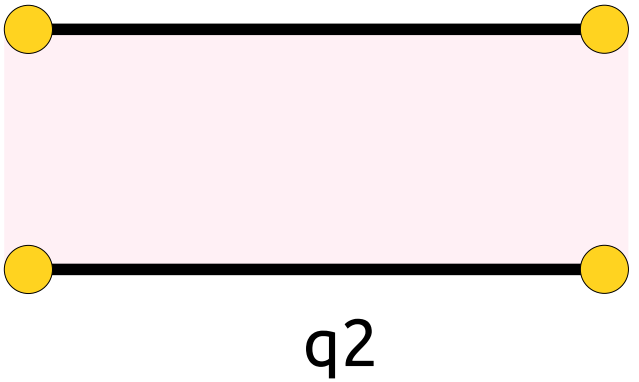


Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```



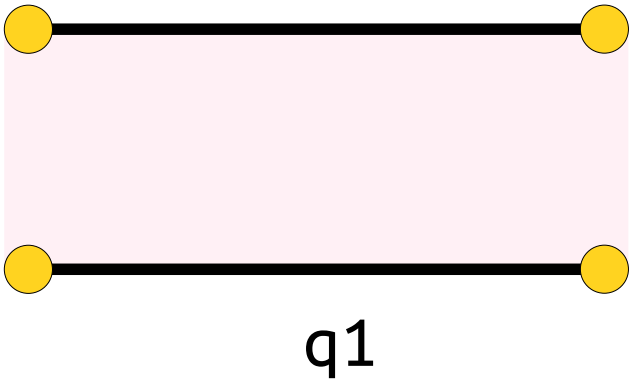


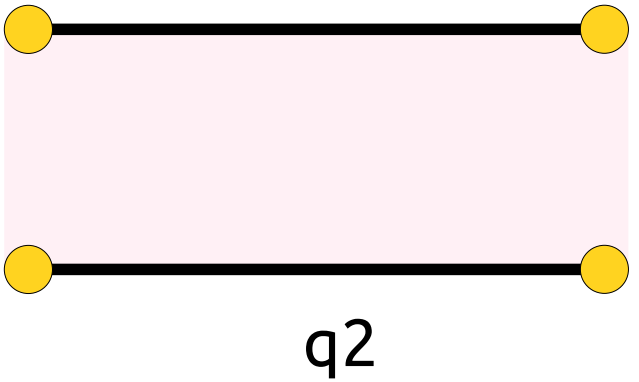
'a'	'b'	'c'

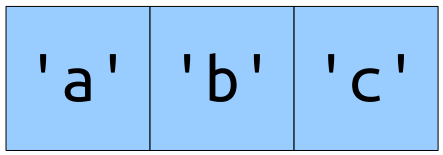
Queue

- What does this code print?

```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

while (!q1.isEmpty()) {
  q2.enqueue(q1.dequeue());
}

while (!q2.isEmpty()) {
  cout << q2.dequeue() << endl;
}
```







An Application: ***Looper***

Loopers

- A ***looper*** is a device that records sound or music, then plays it back over and over again (in a loop).
- These things are way too much fun, *especially* if you're not a very good musician.
- Let's make a simple looper using a `Queue`.

Building our Looper

- Our looper will read data files like the one shown to the left.
- Each line consists of the name of a sound file to play, along with how many milliseconds to play that sound for.
- We'll store each line using the `SoundClip` type, which is defined in our C++ file.

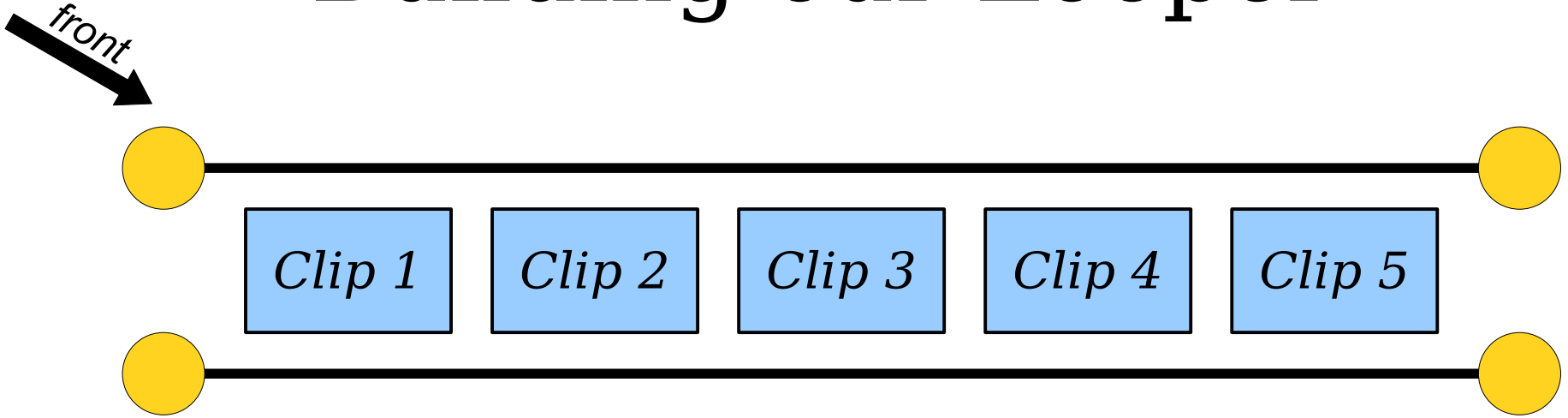
```
G2.wav 690
G2.wav 230
Bb2.wav 230
G2.wav 460
G2.wav 460
G2.wav 460
G2.wav 230
Bb2.wav 230
G2.wav 230
F2.wav 460
```

Building our Looper

Building our Looper

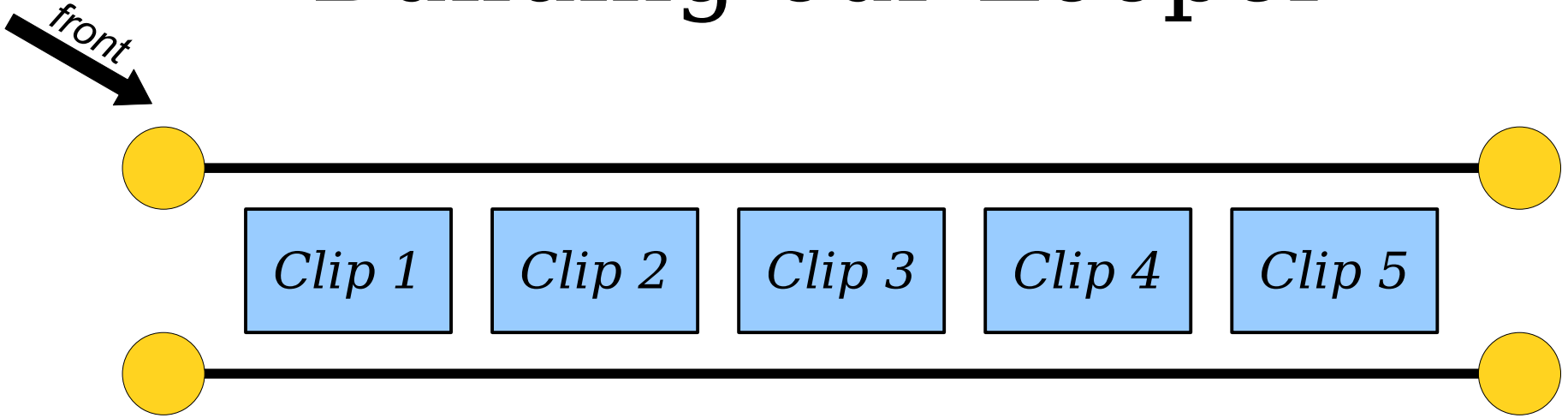
```
Queue<SoundClip> loop = loadLoop(/* … */);
```

Building our Looper



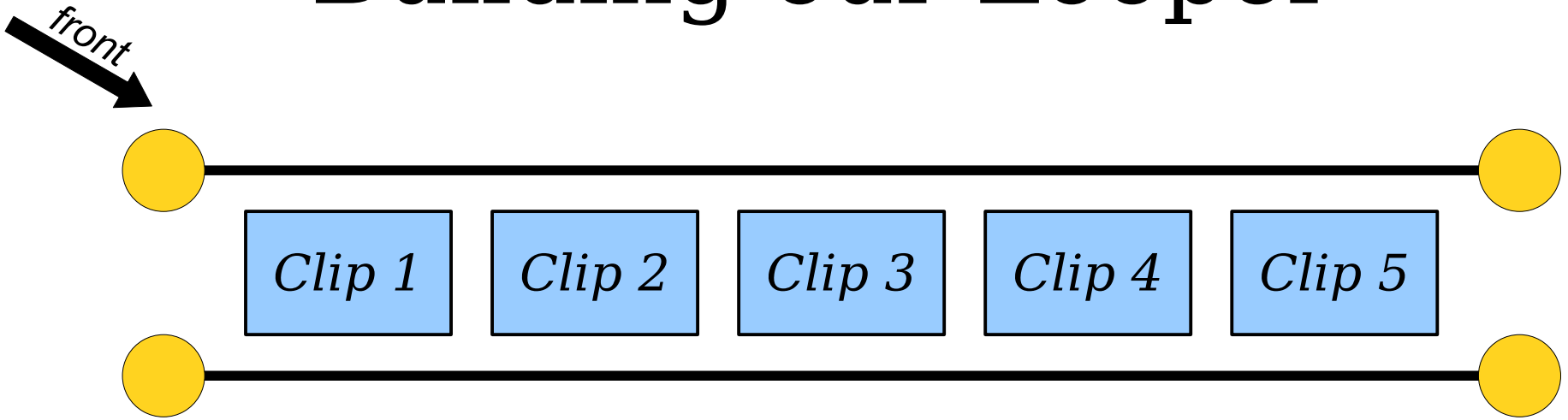

```
Queue<SoundClip> loop = loadLoop(/* … */);
```

Building our Looper



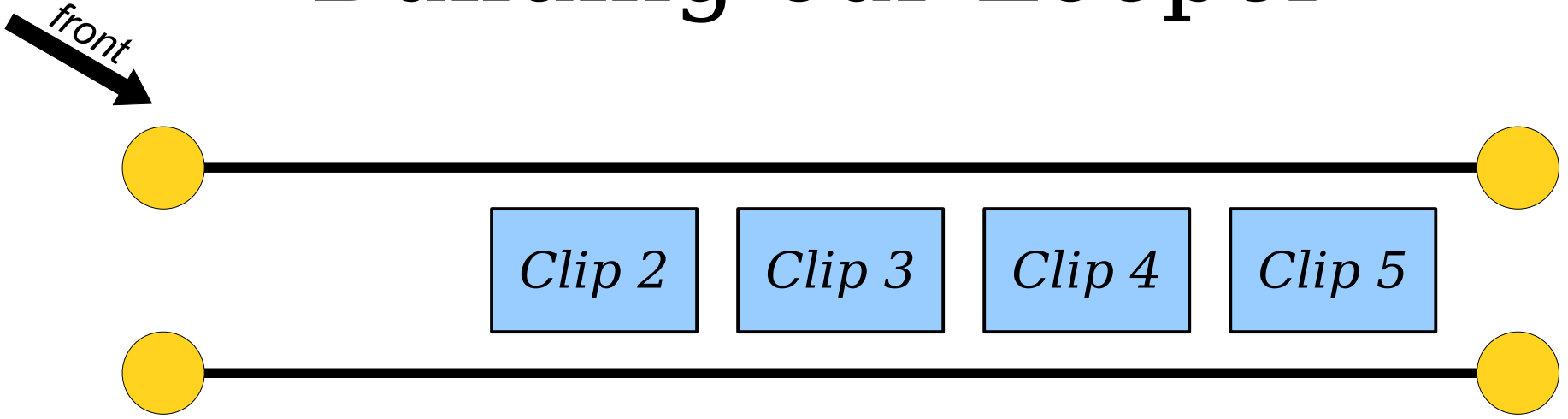

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {



}
```

Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();



}
```

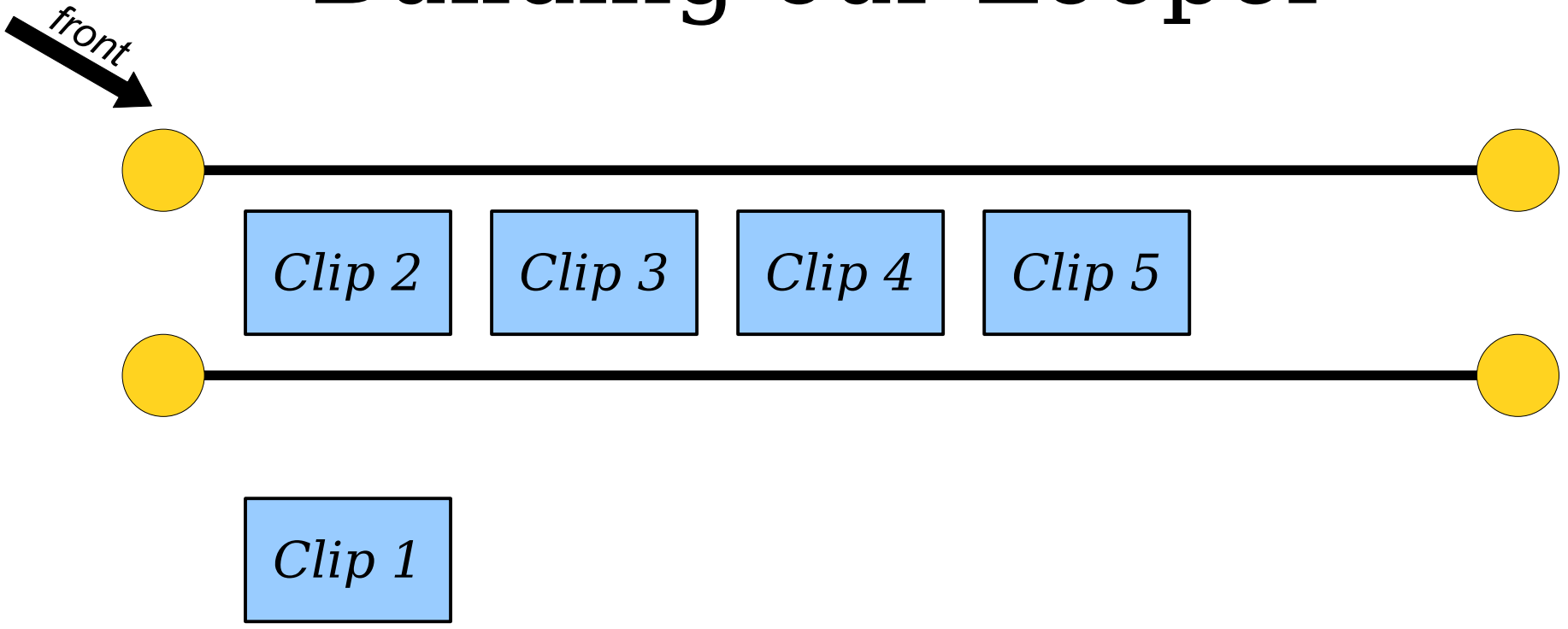
Building our Looper



Clip 1

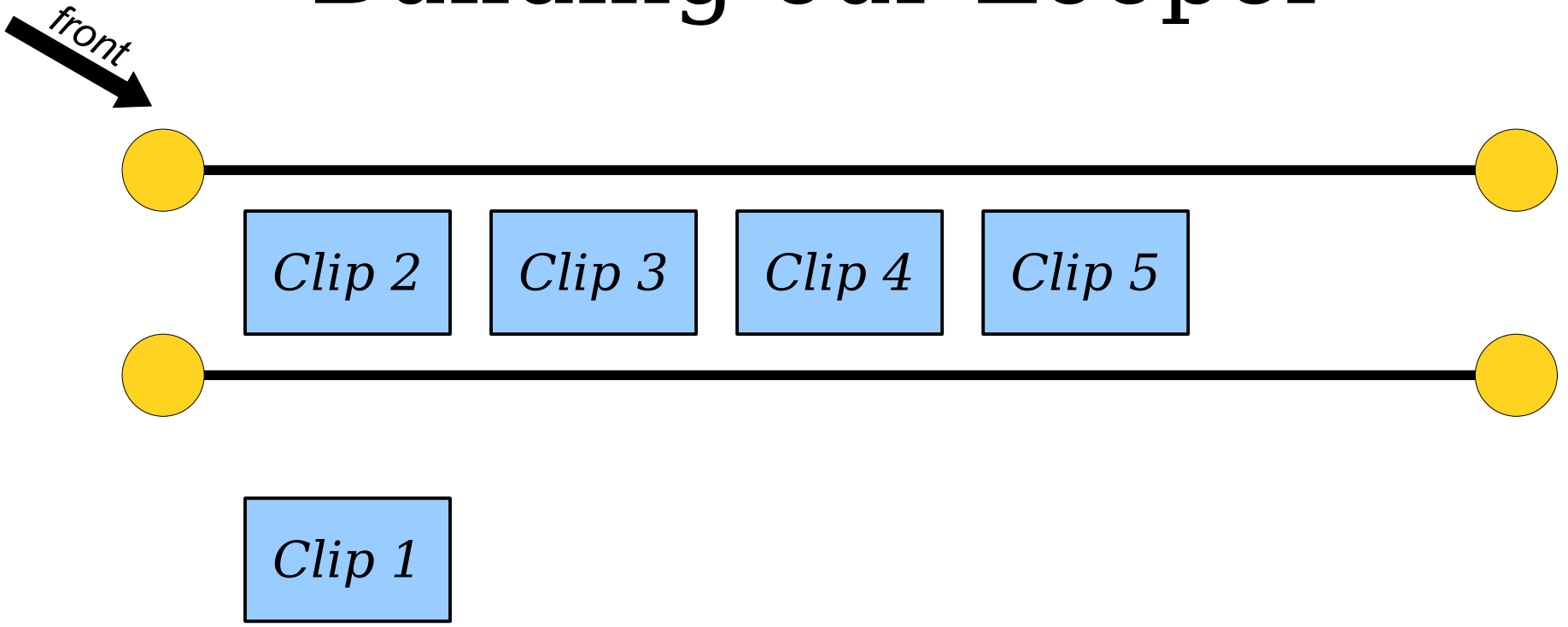
```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();



}
```

Building our Looper



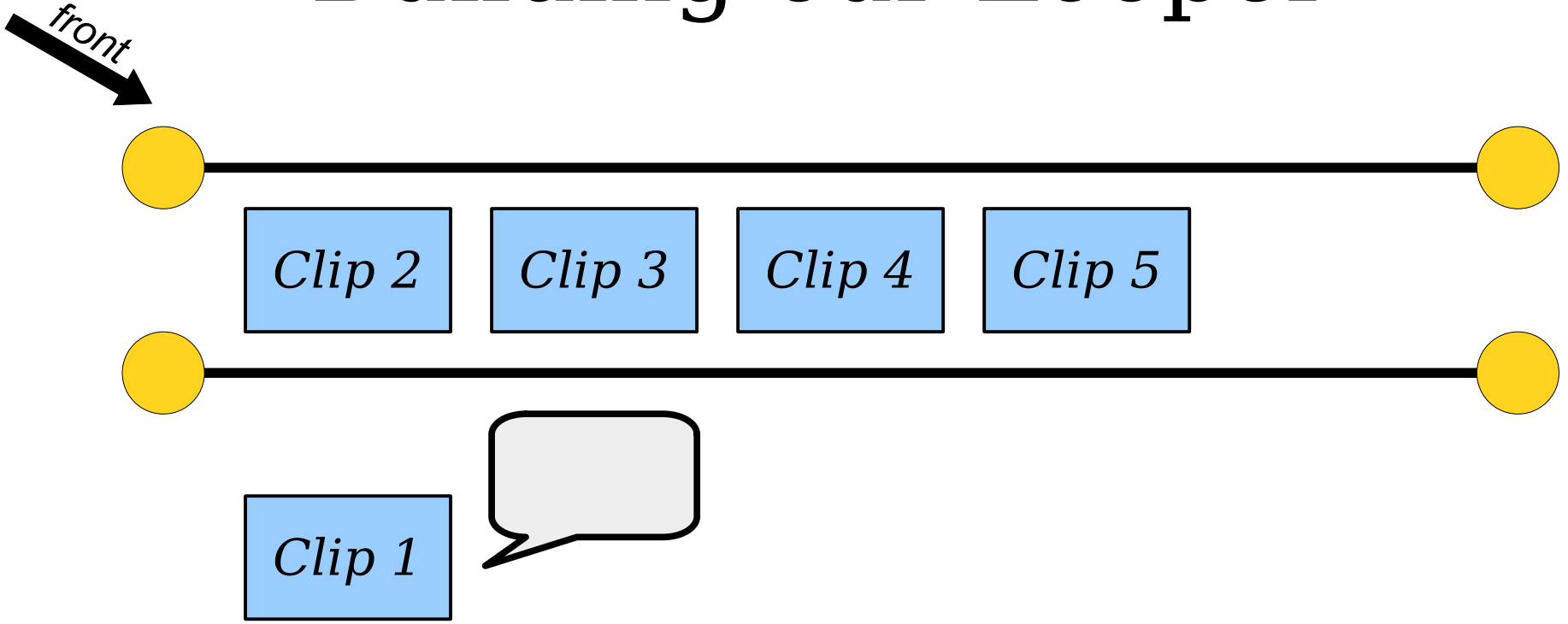

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();



}
```

Building our Looper



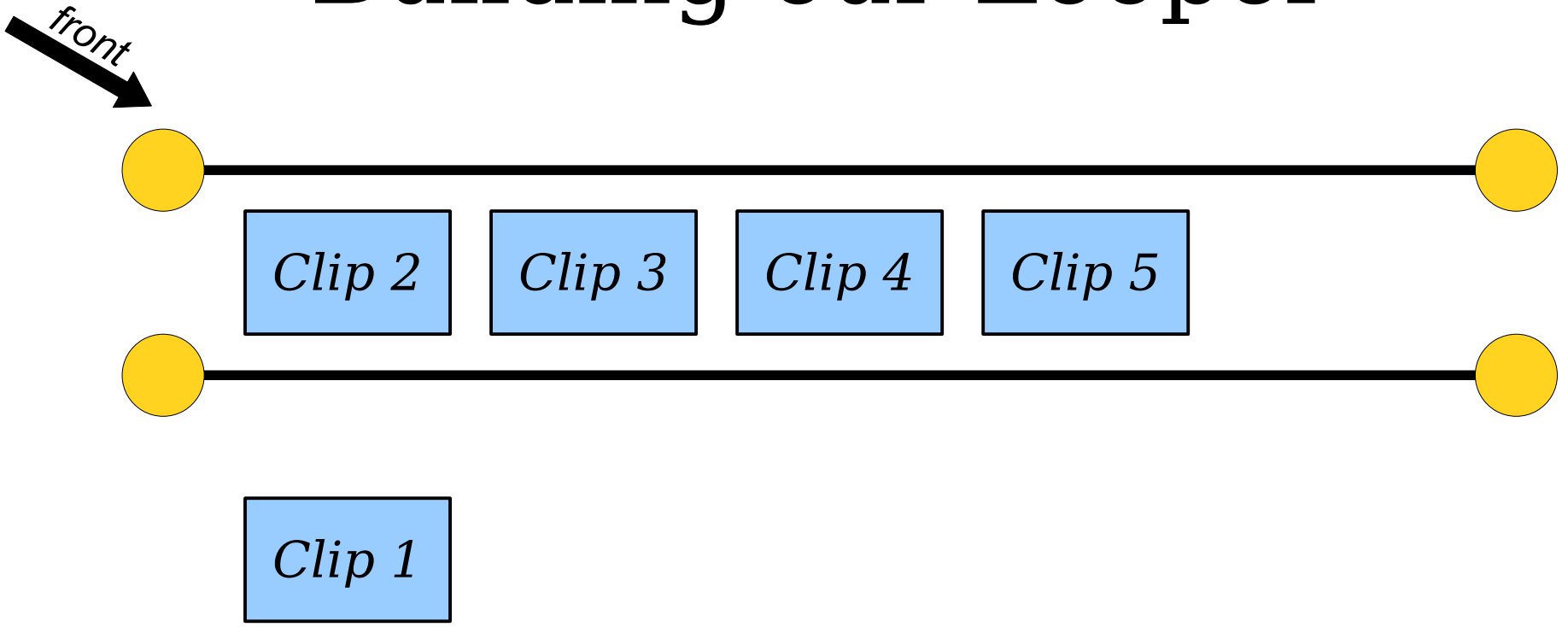

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

Building our Looper



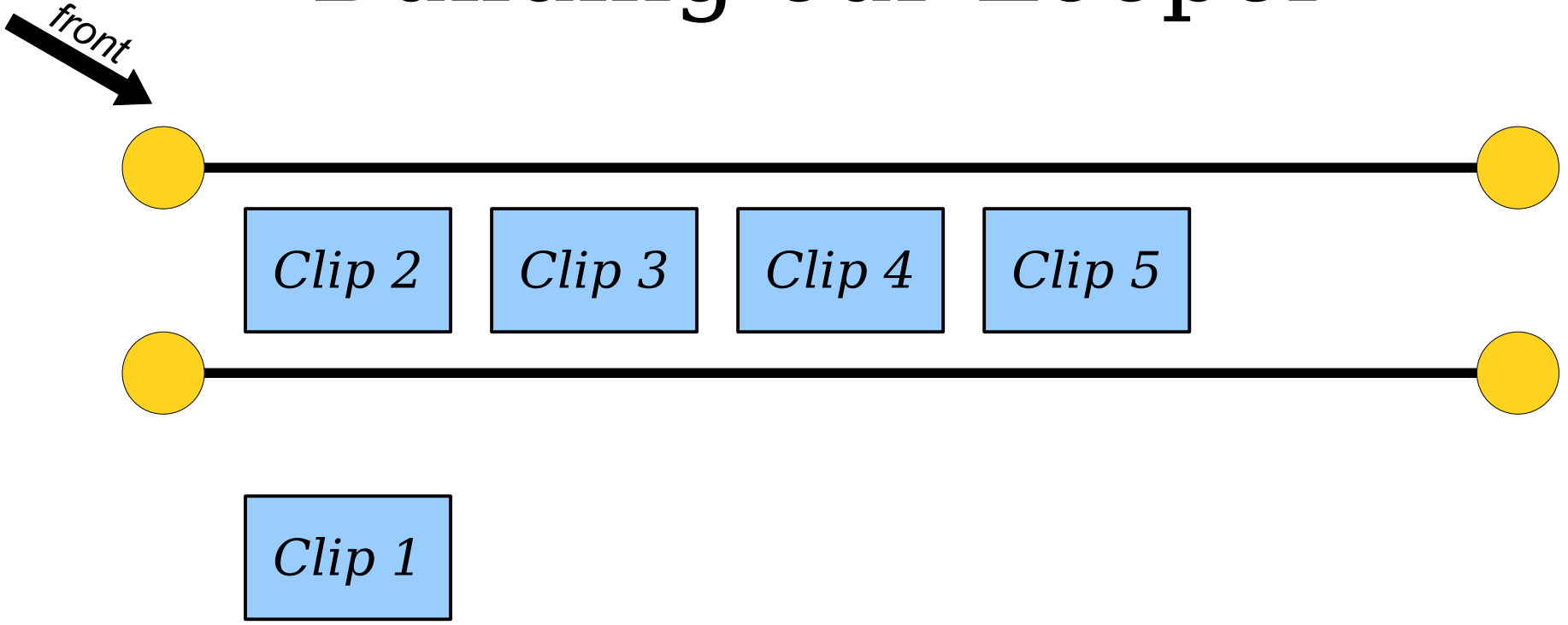

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

Building our Looper



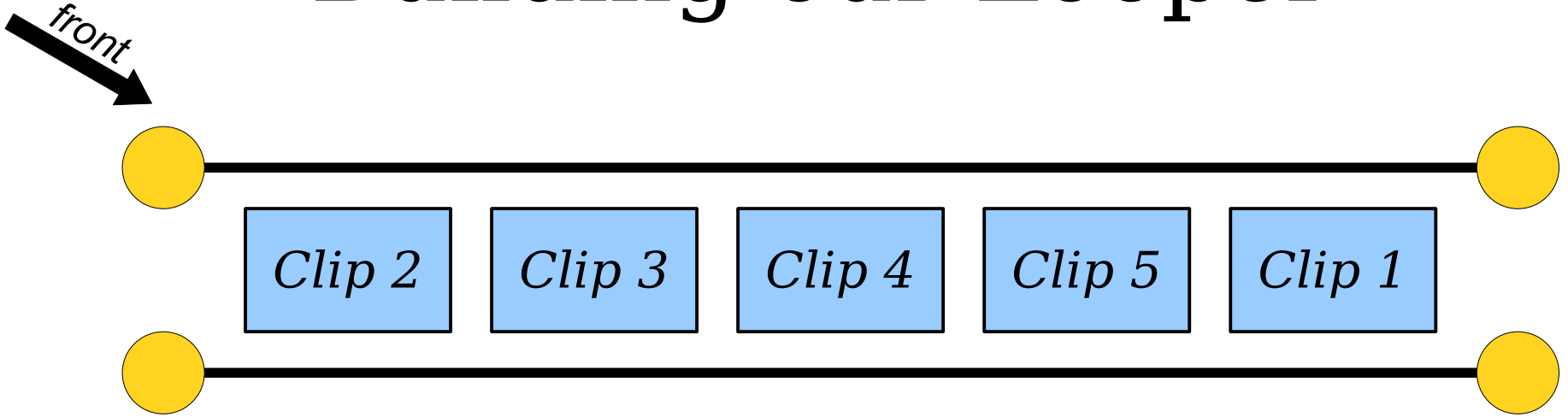

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

Building our Looper



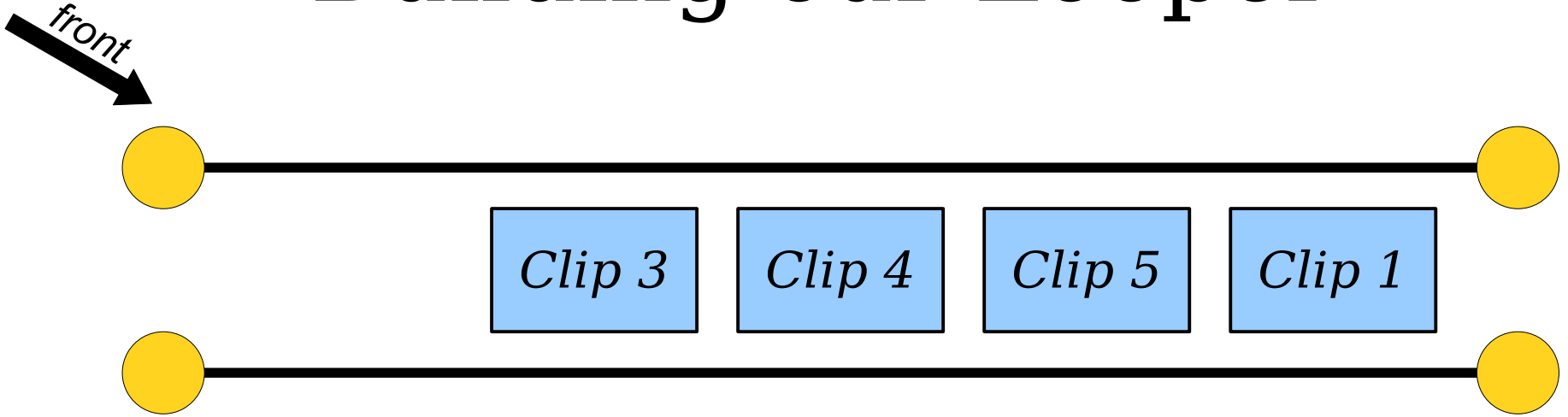

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

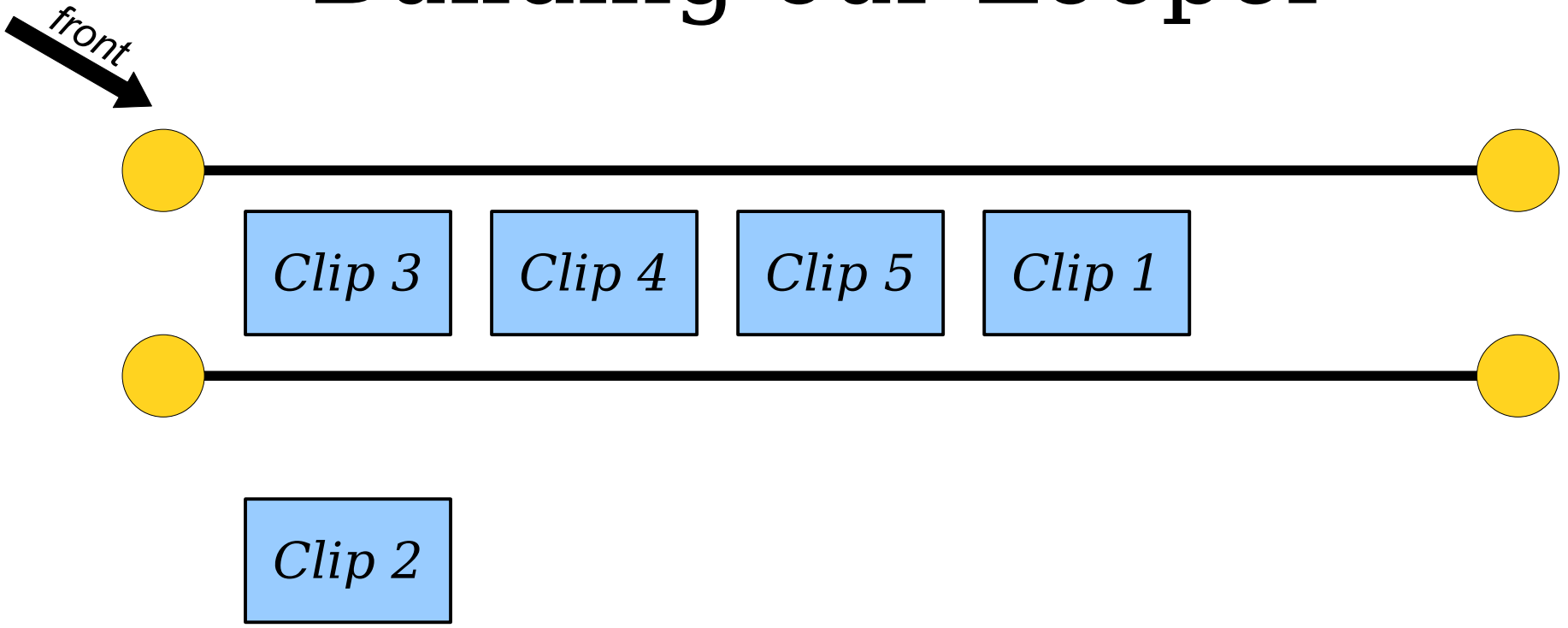
Building our Looper



Clip 2

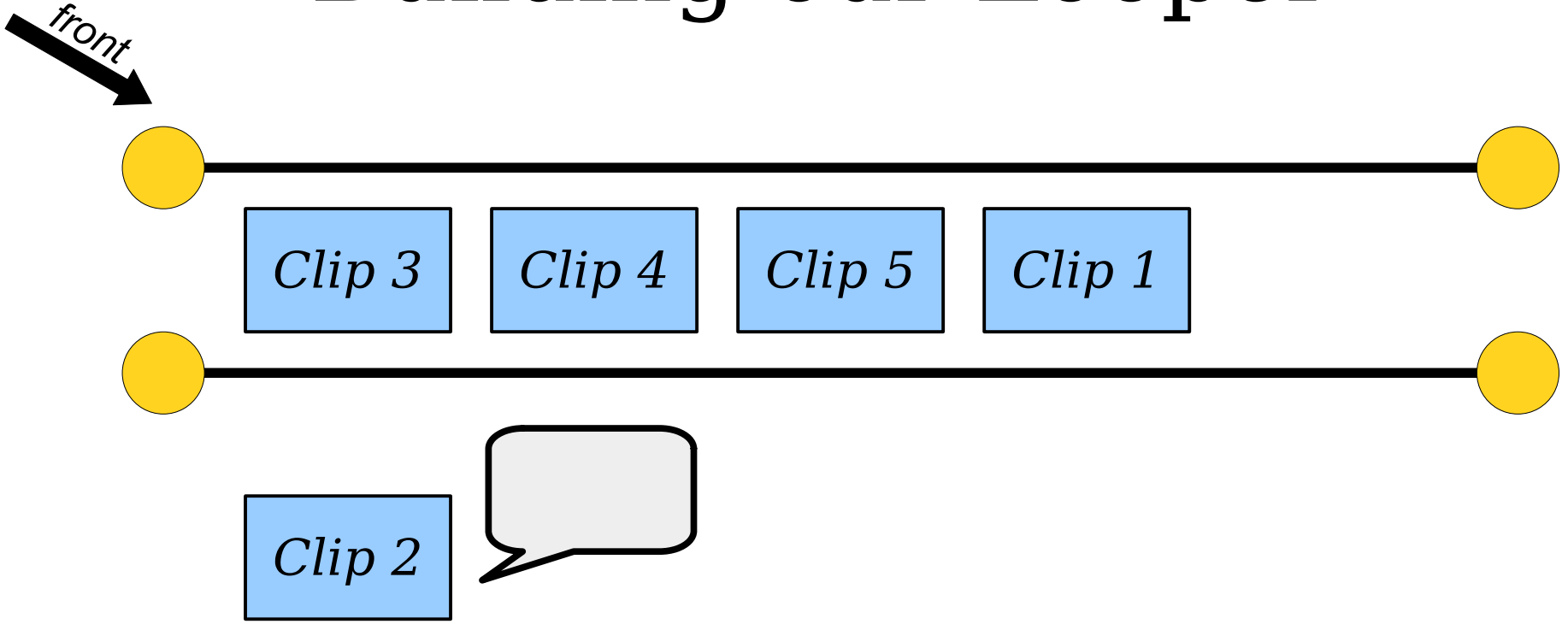
```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



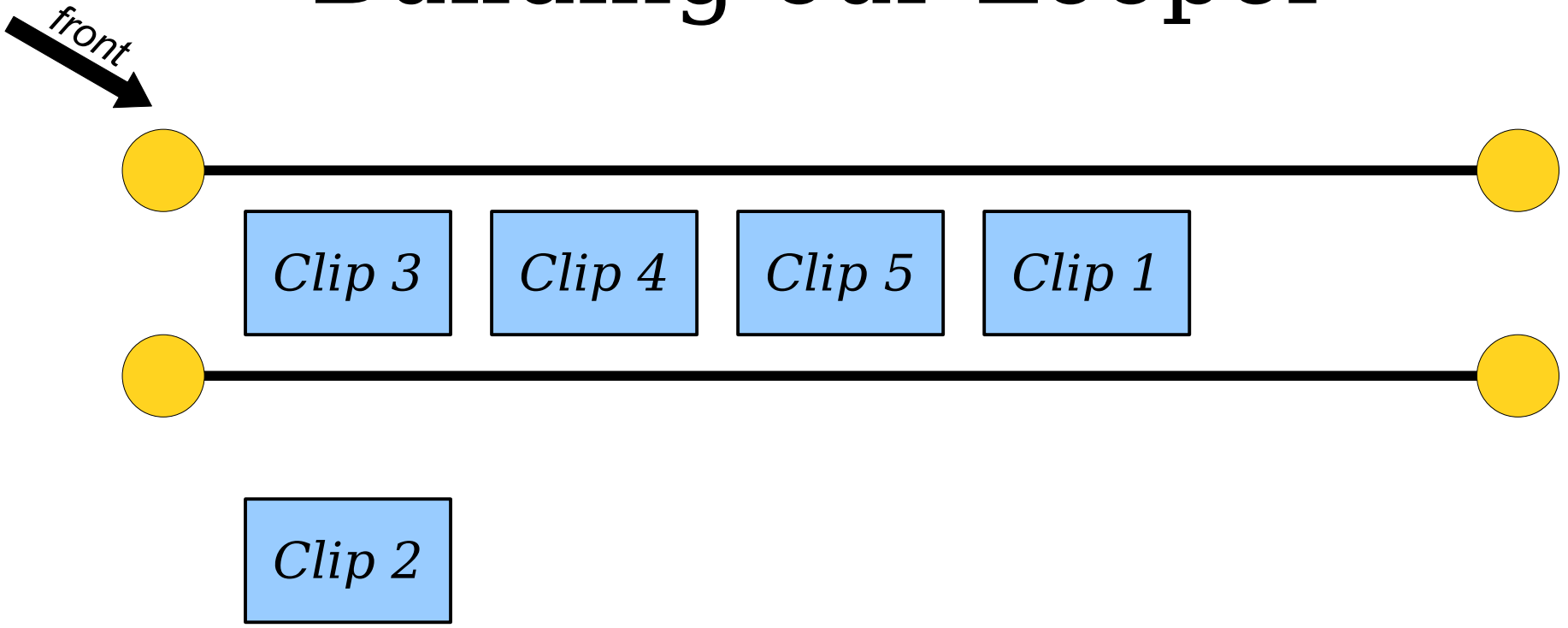

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



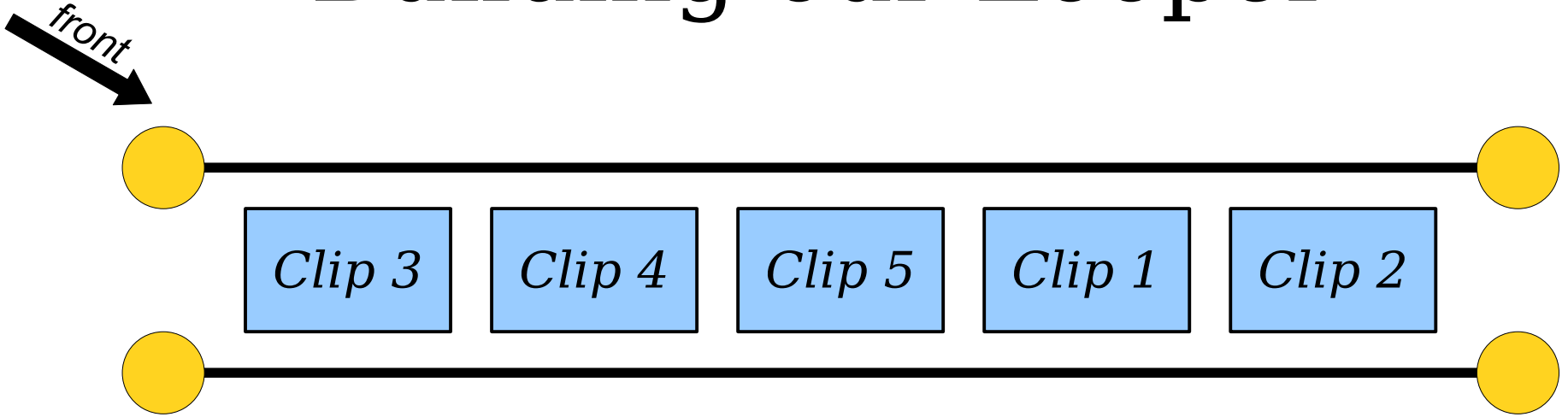

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



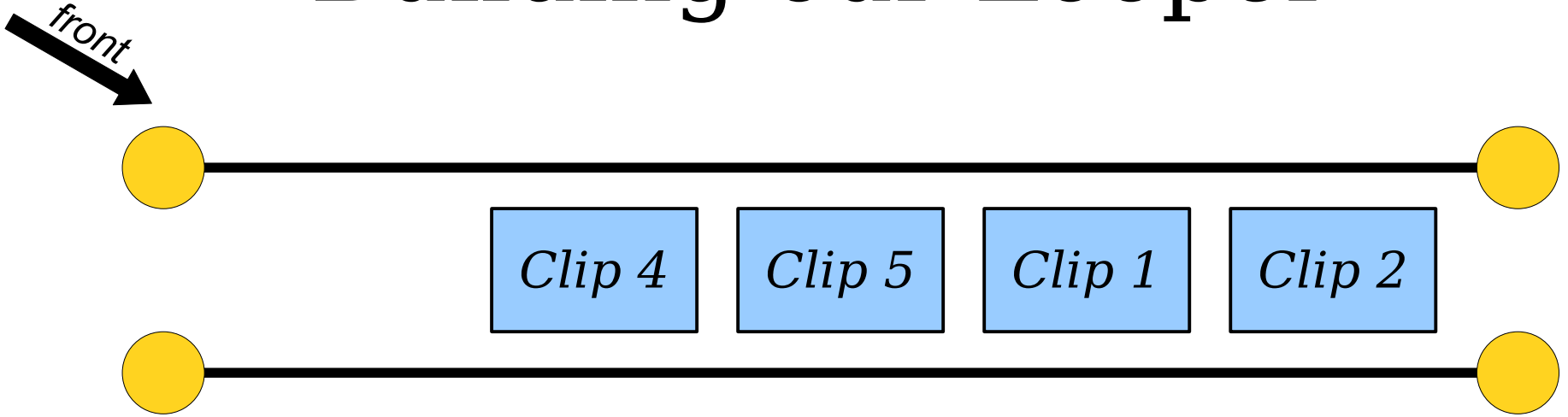

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

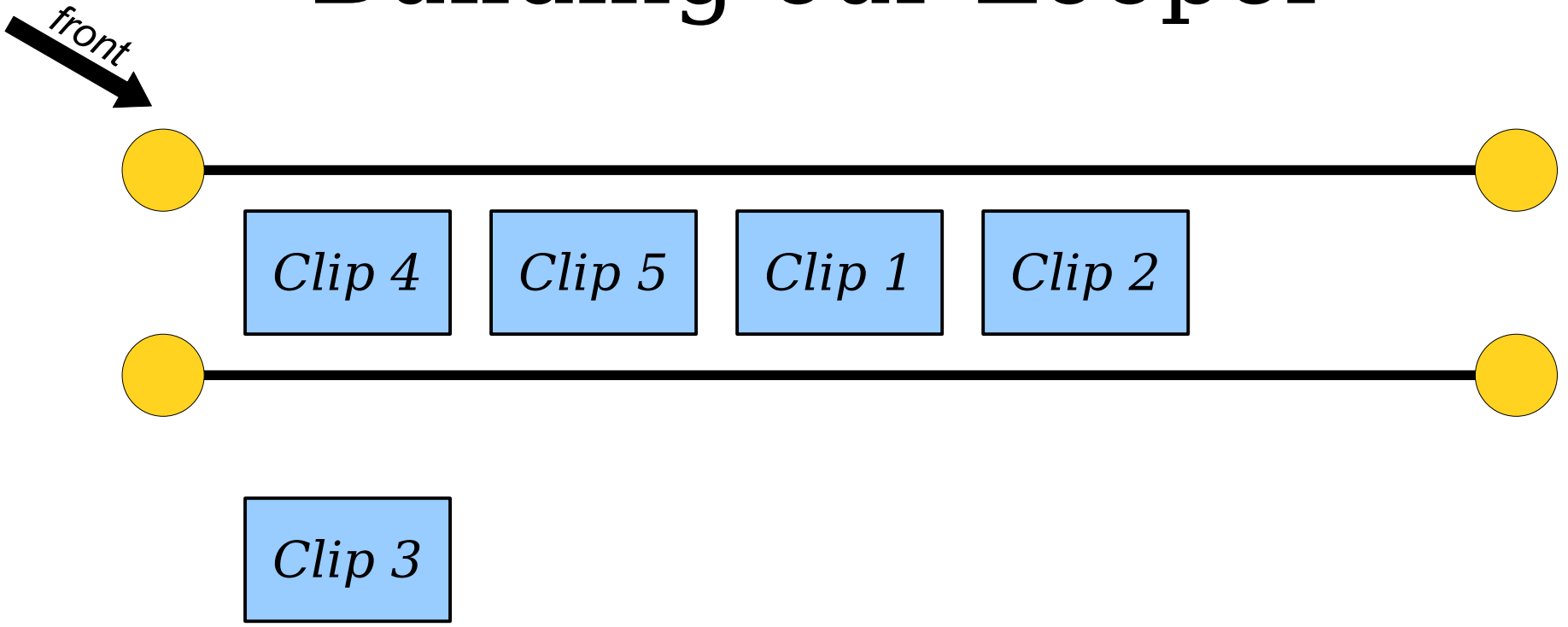
Building our Looper



Clip 3

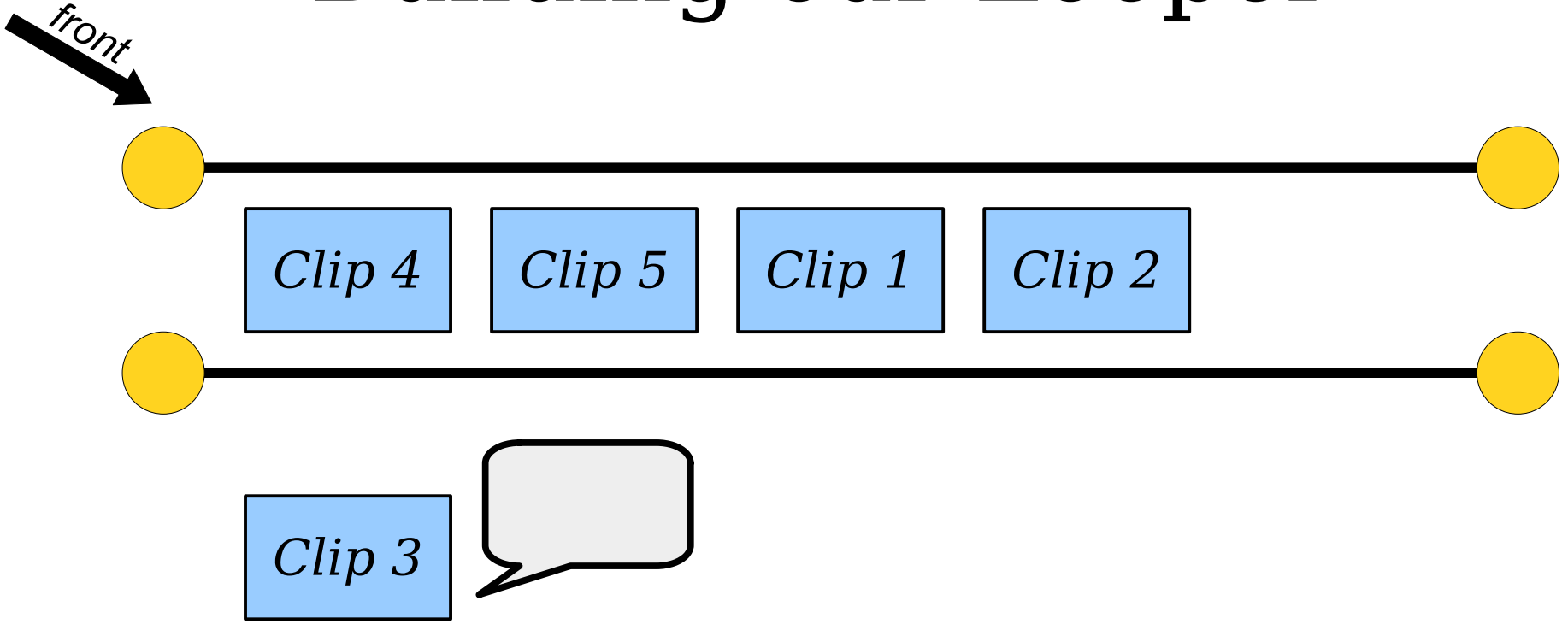
```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



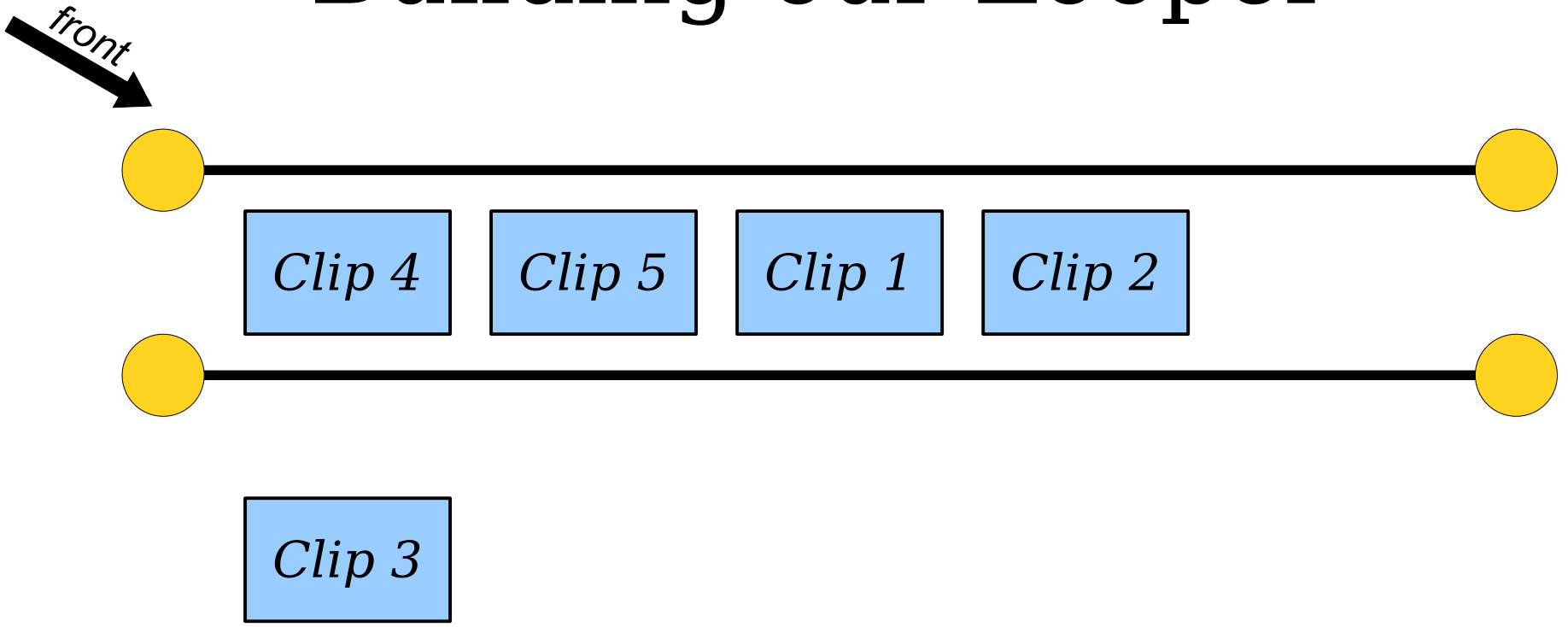

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



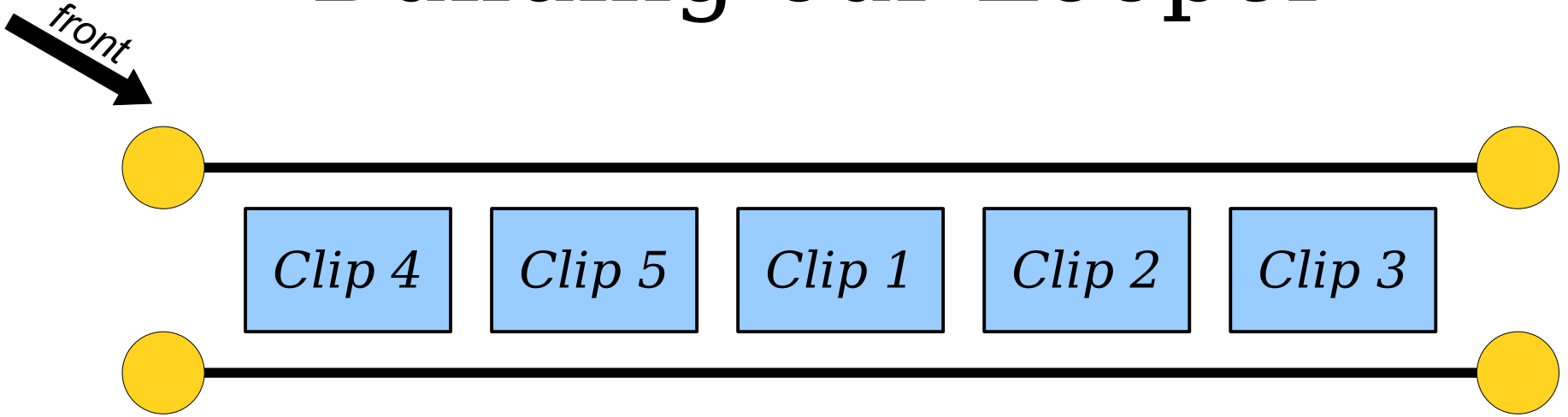

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper




```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Enjoying Our Looper

Feeling musical? Want to contribute a loop for the next iteration of CS106B? Send me your .loop file and we'll add it to our collection!

Changing our Looper

Changing our Looper

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Changing our Looper

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

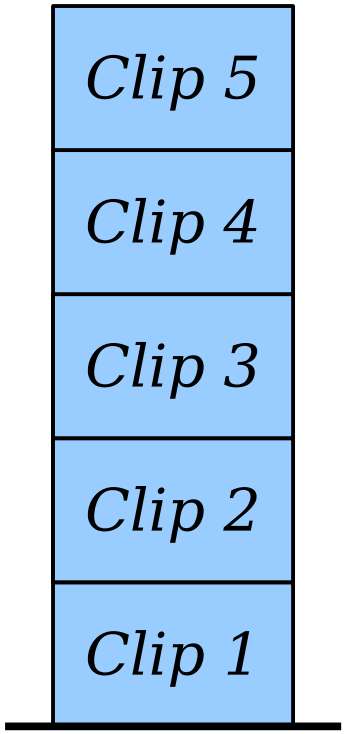
Changing our Looper

What are you going to hear when we use this version of the looper?

Answer at
https://cs106b.stanford.edu/pollev

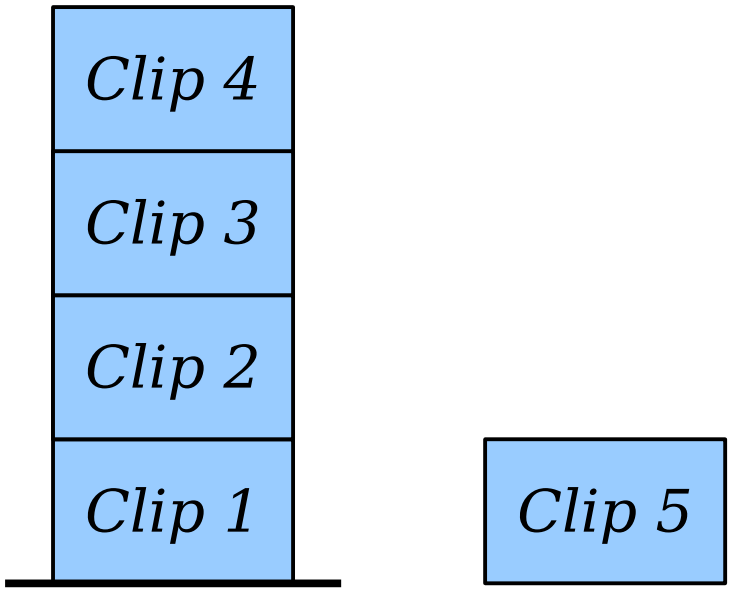
```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



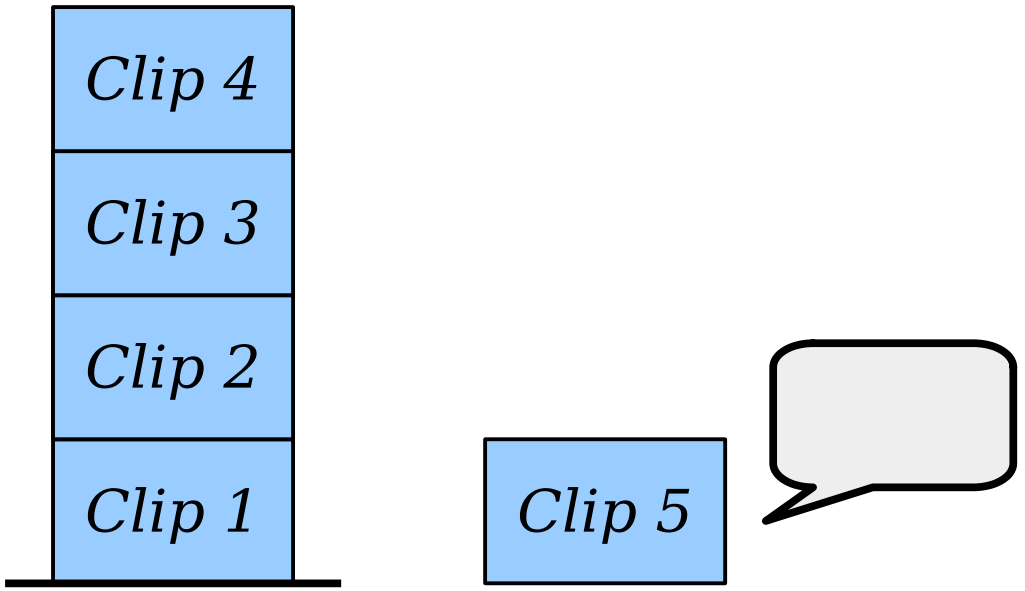

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



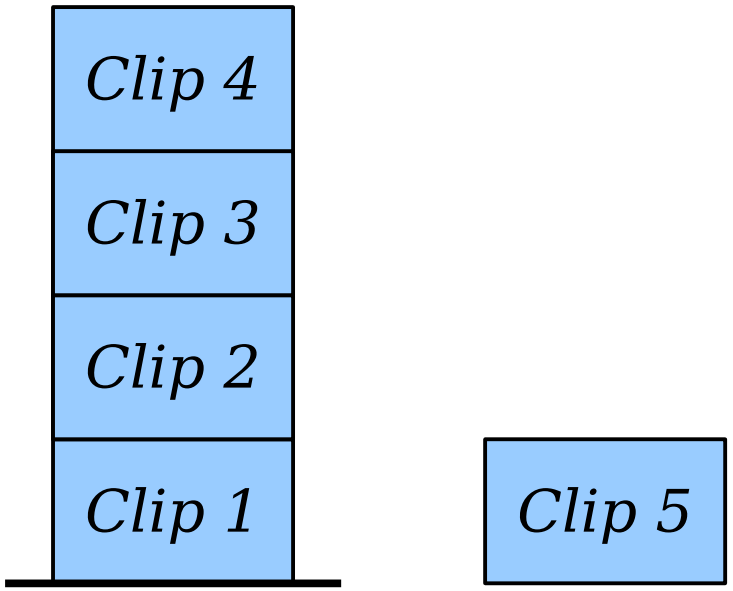

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



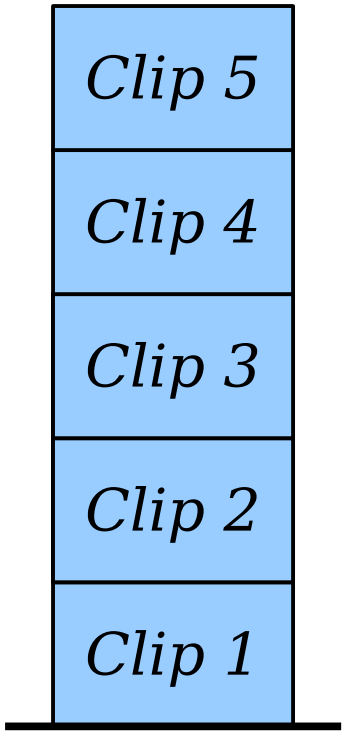

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



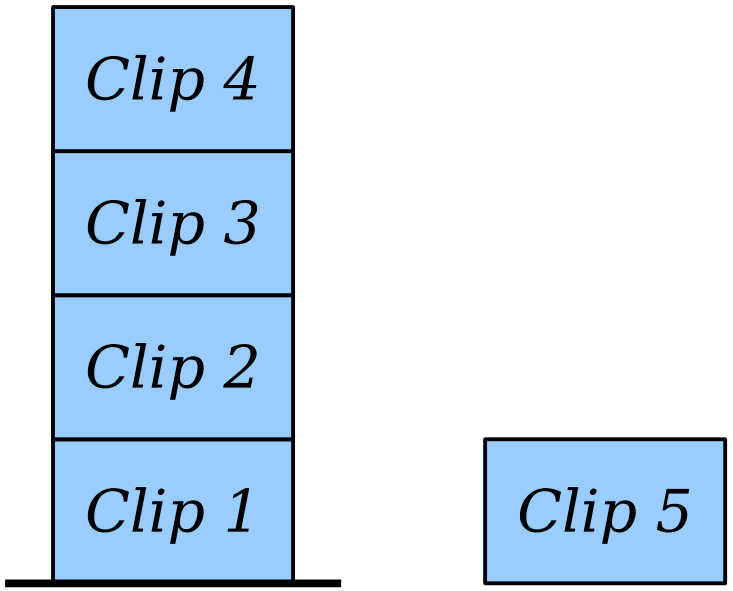

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



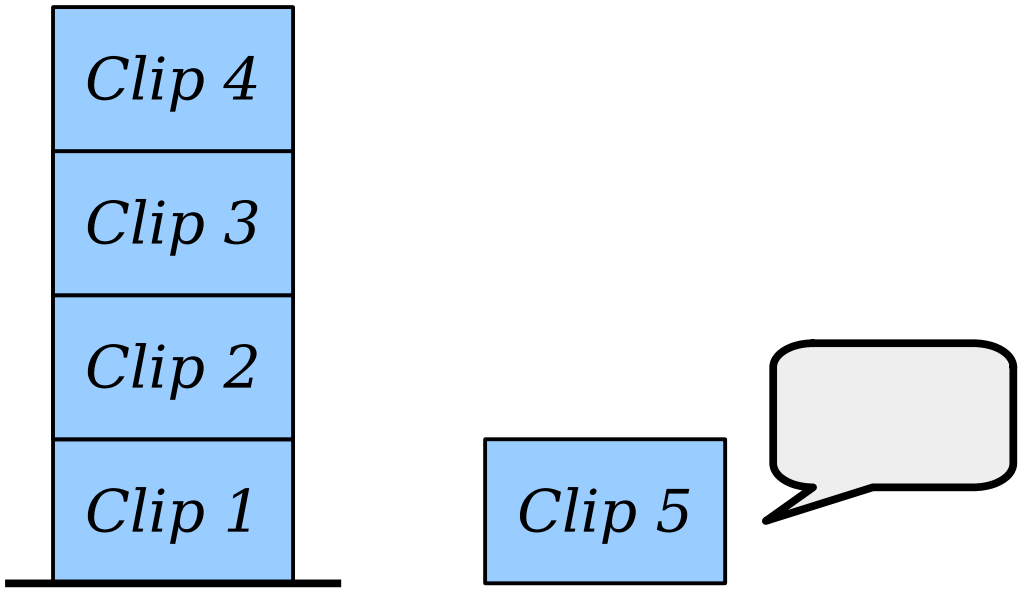

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



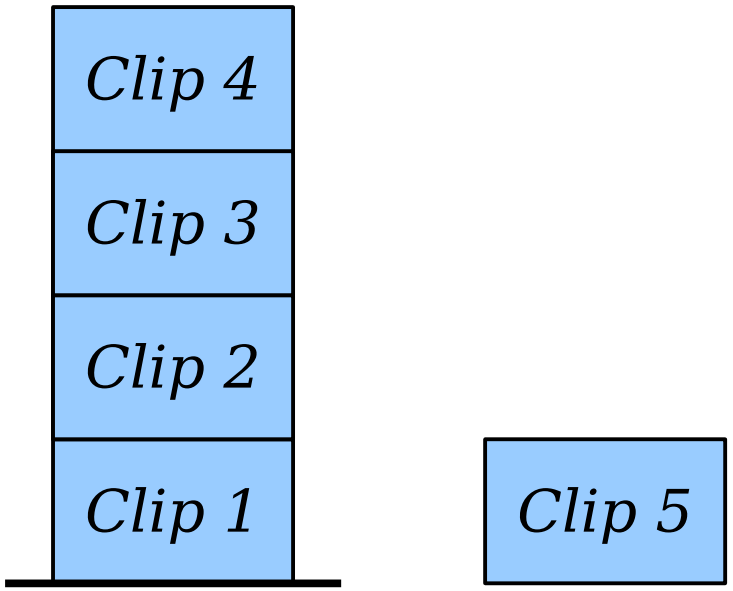

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



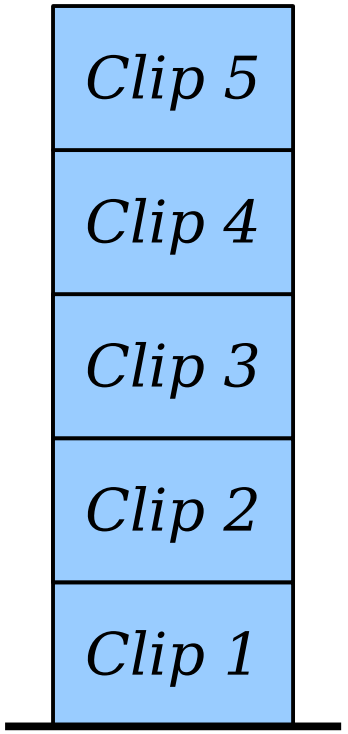

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper




```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper




```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Your Action Items

- ***Read Chapter 5.2 and 5.3.***
  - These sections cover more about the `Stack` and `Queue` type, and they're great resources to check out.
- ***Start Assignment 2.***
  - To follow our suggested timetable, start working on Rosetta Stone and make good progress on it by Monday.

Next Time (Virtually!)

- ***Thinking Recursively***
  - More elaborate recursive functions.
- ***Recursive Graphics***
  - Drawing intricate and beautiful figures with very little code.