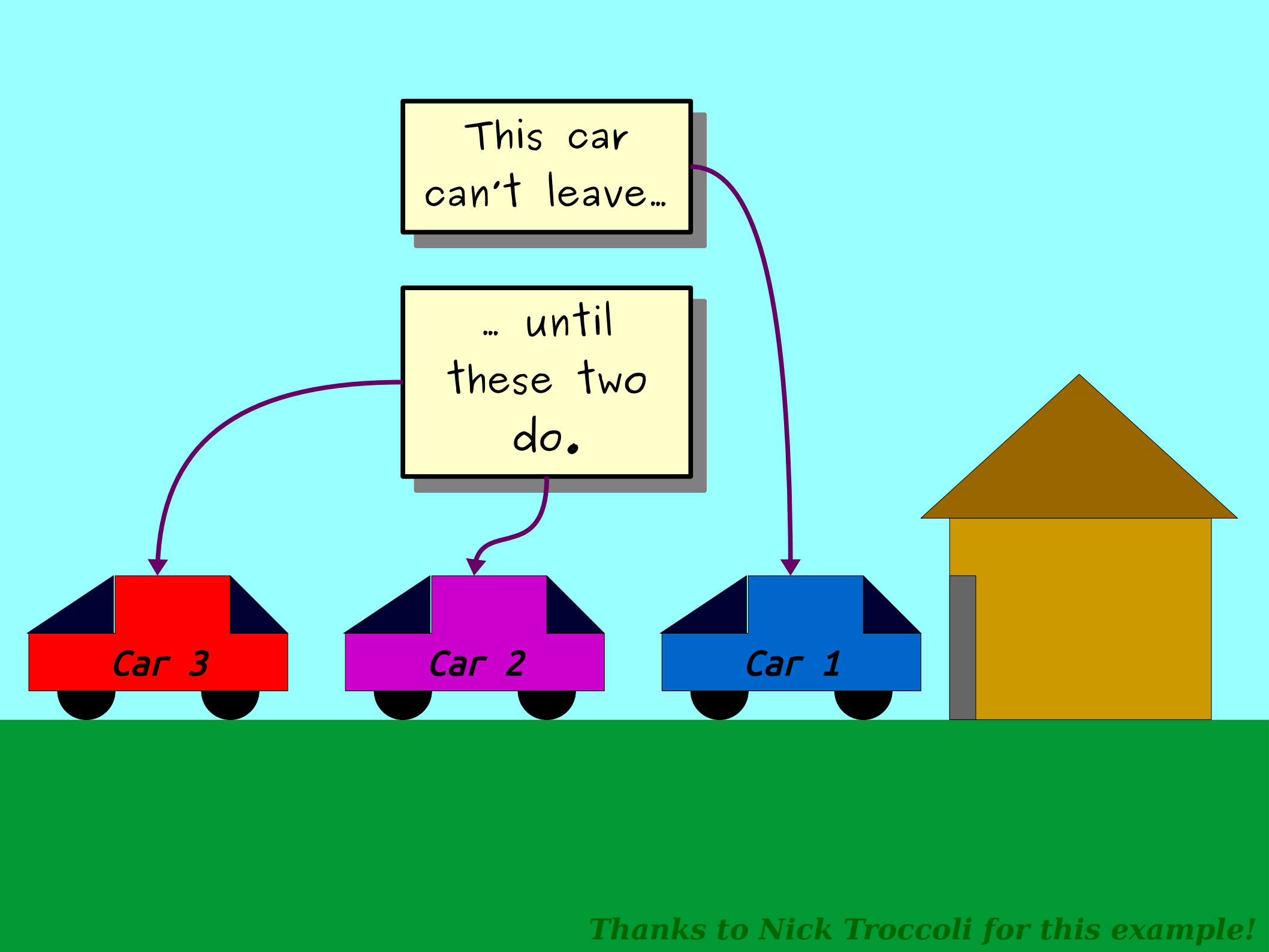
Collections

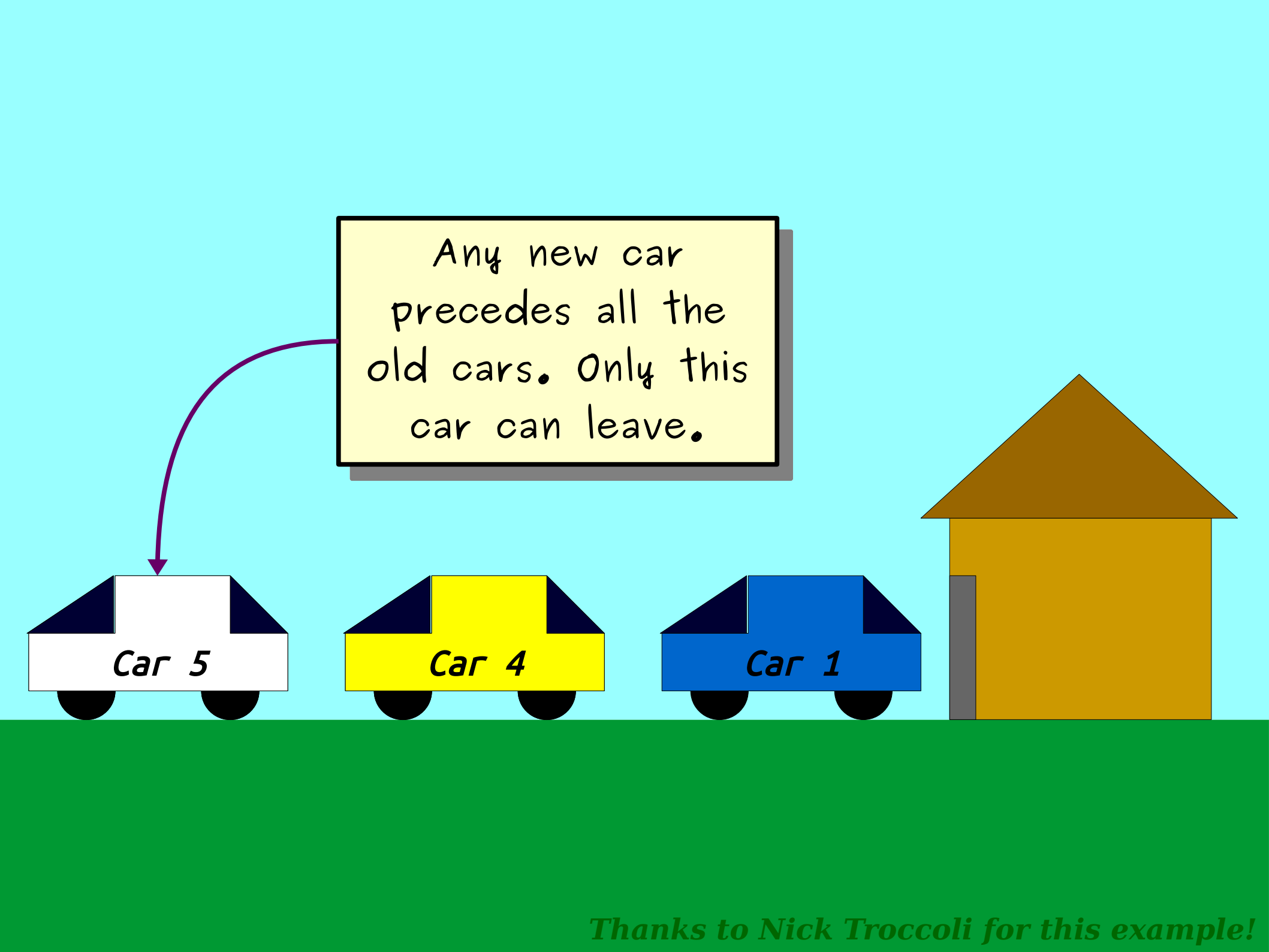
Part Three

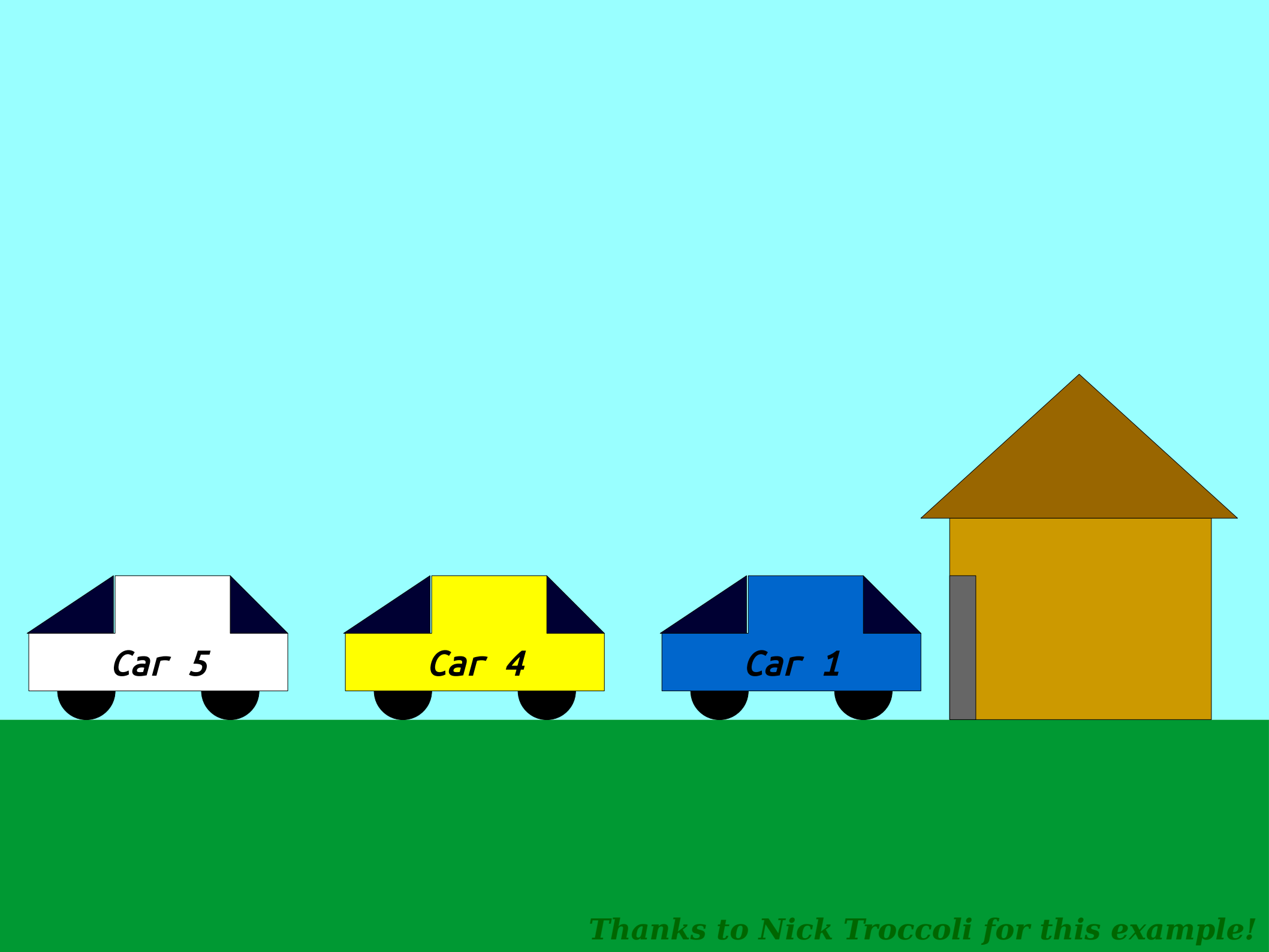
Outline for Today

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

Stack


This car can't leave…
… until these two do.
Car 3
Car 2
Car 1
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

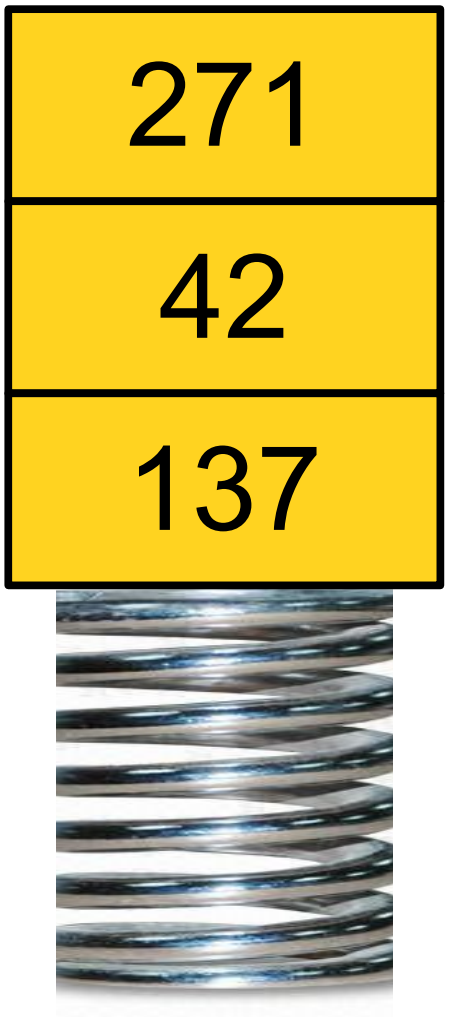
- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

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.

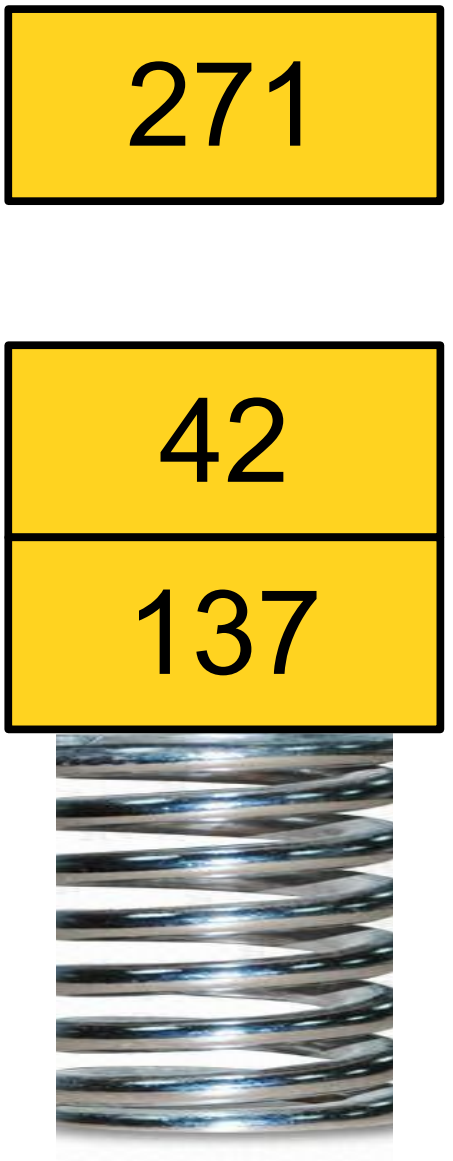
Stack

- 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

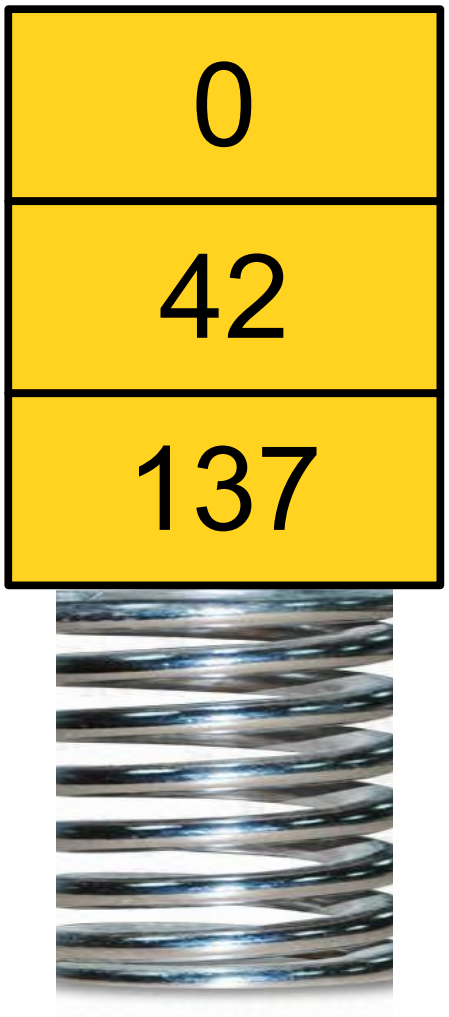
- A **`Stack`** is a data structure representing a stack of things.
- Objects can be ***pushed*** on top of the stack or ***popped*** from the top of the stack.

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.

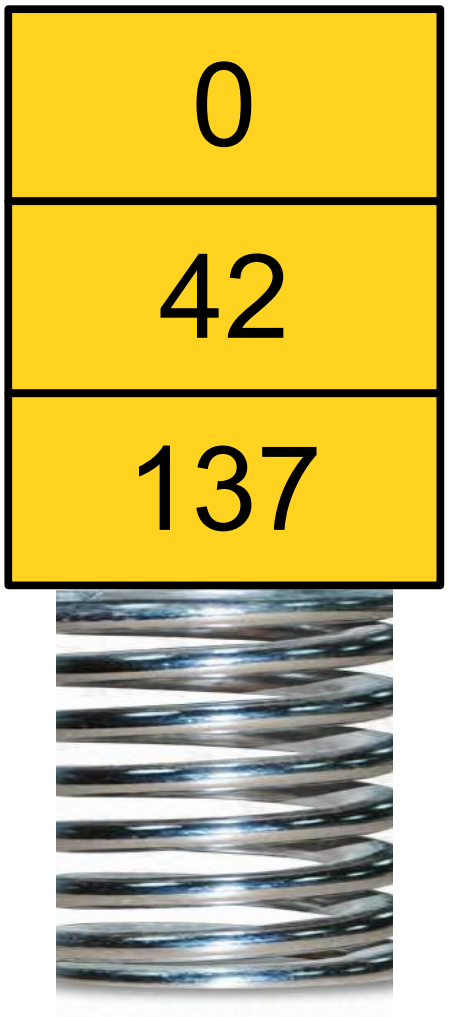
Stack

- 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://pollev.com/cs106bwin23

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

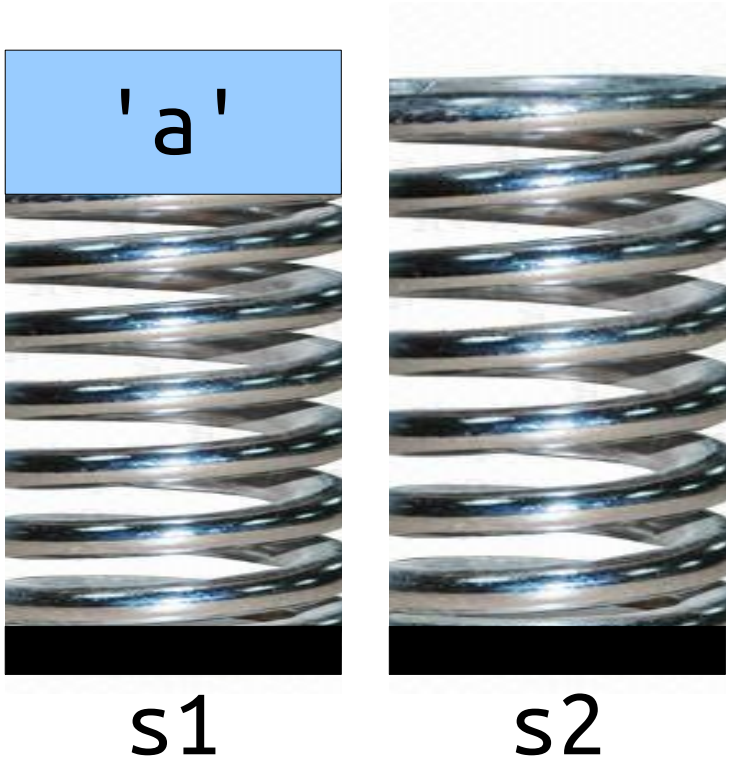
What does this code print?

```
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?

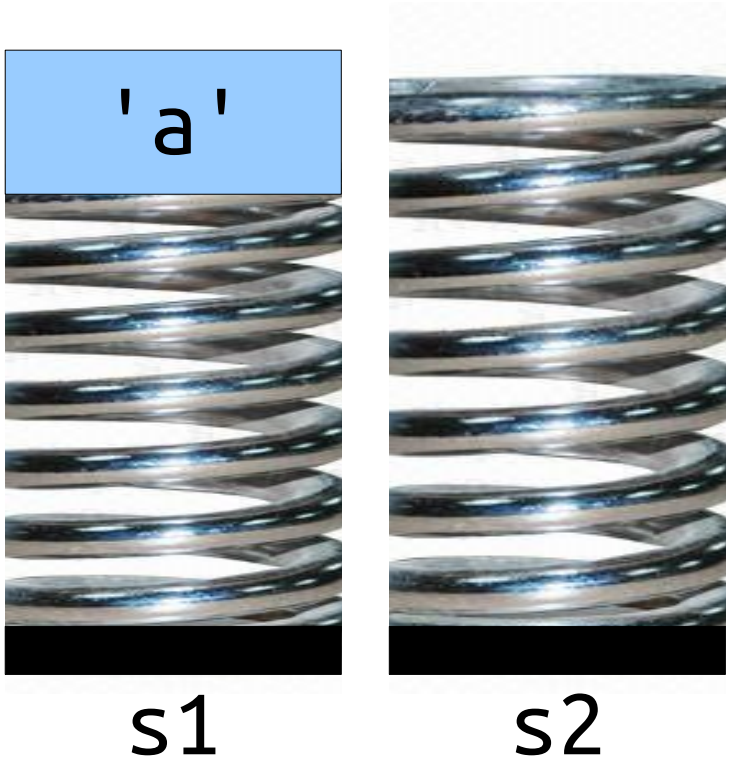
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

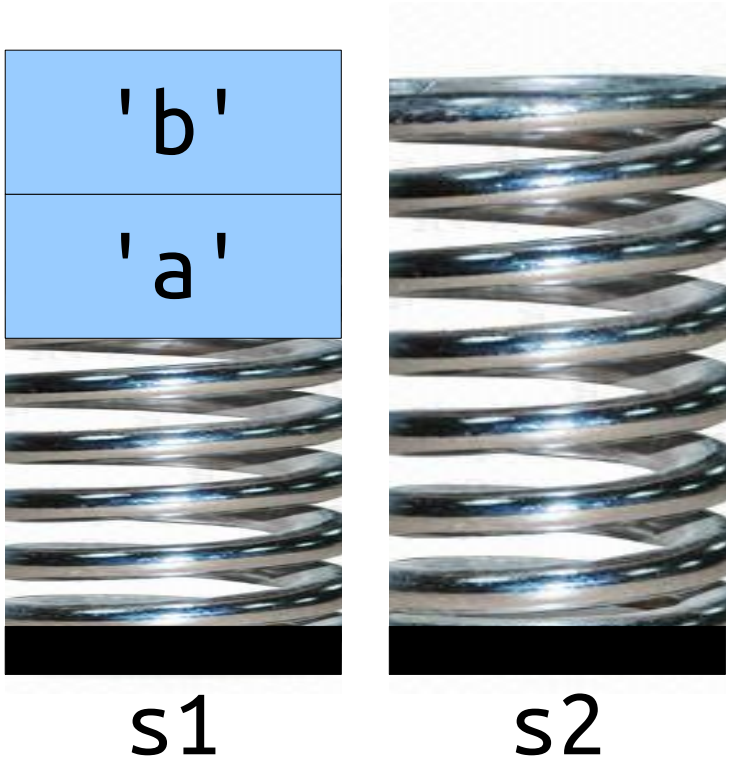
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

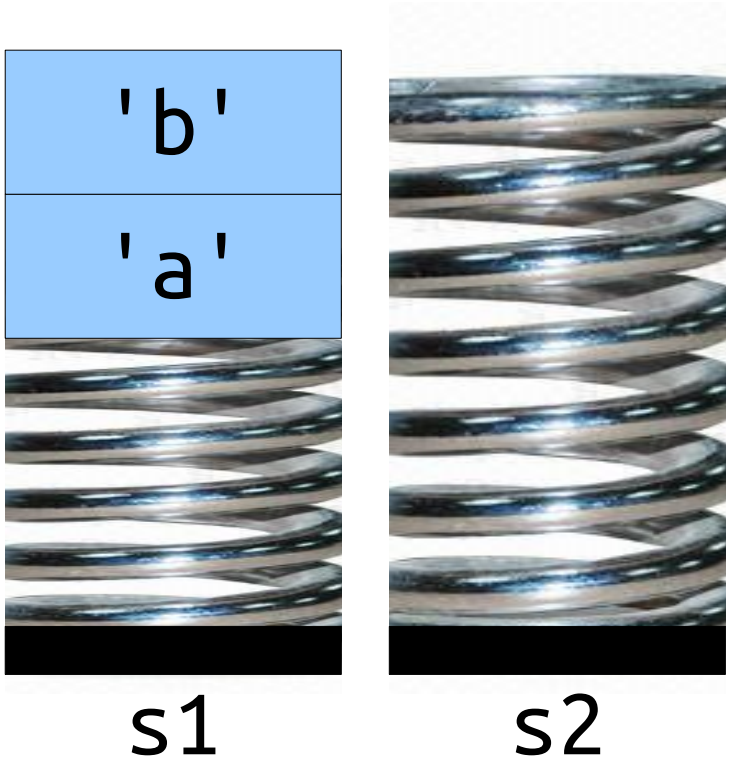
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

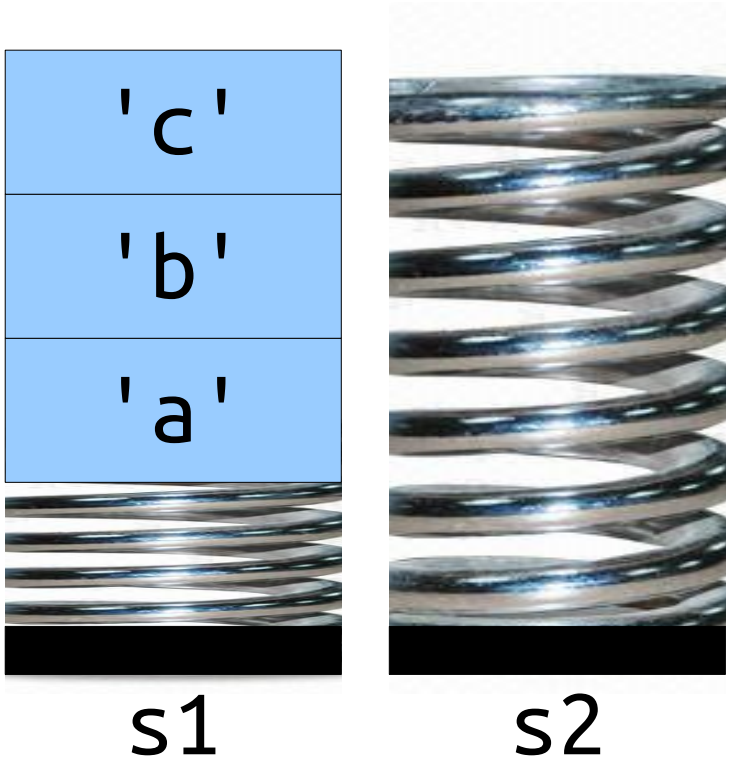
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

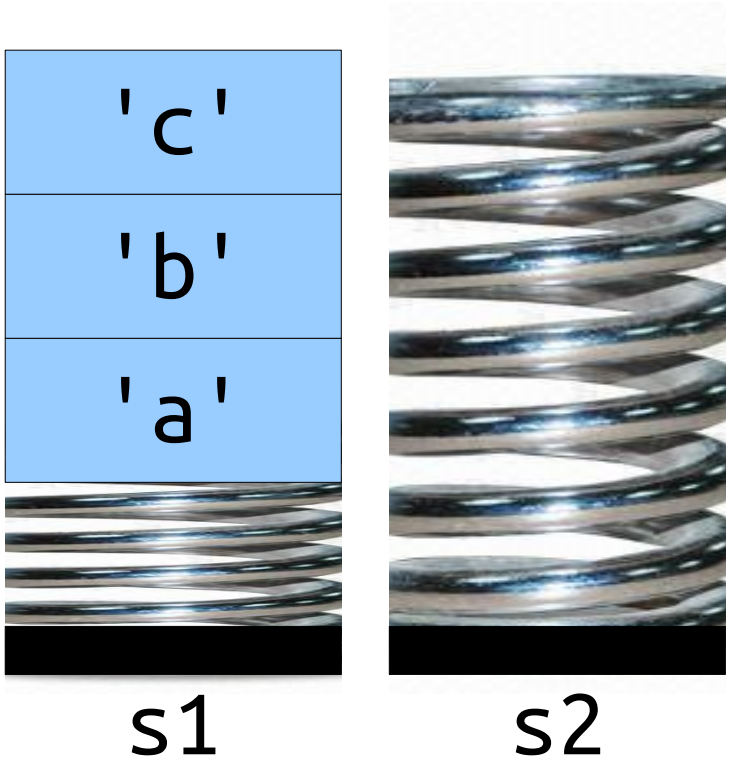
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

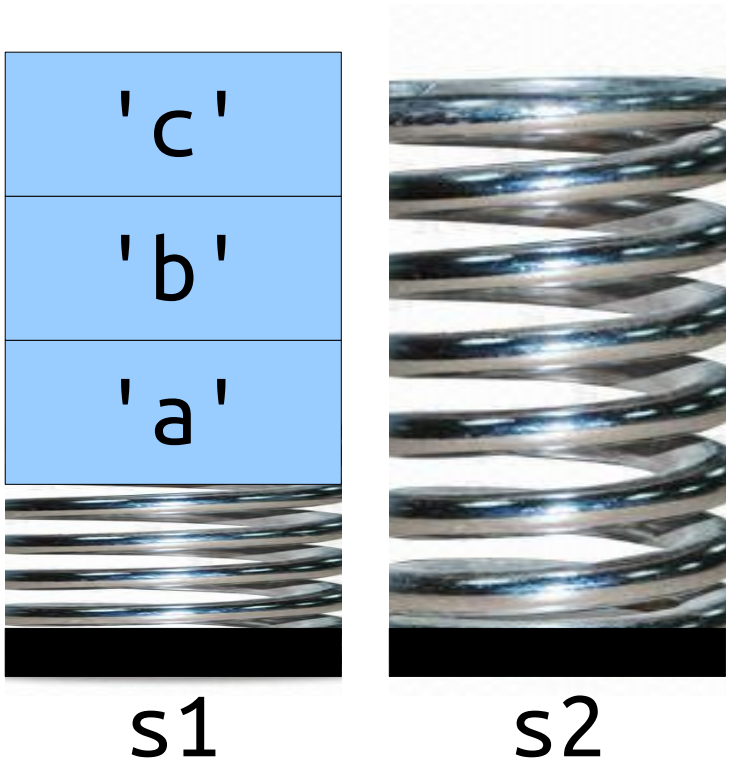
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

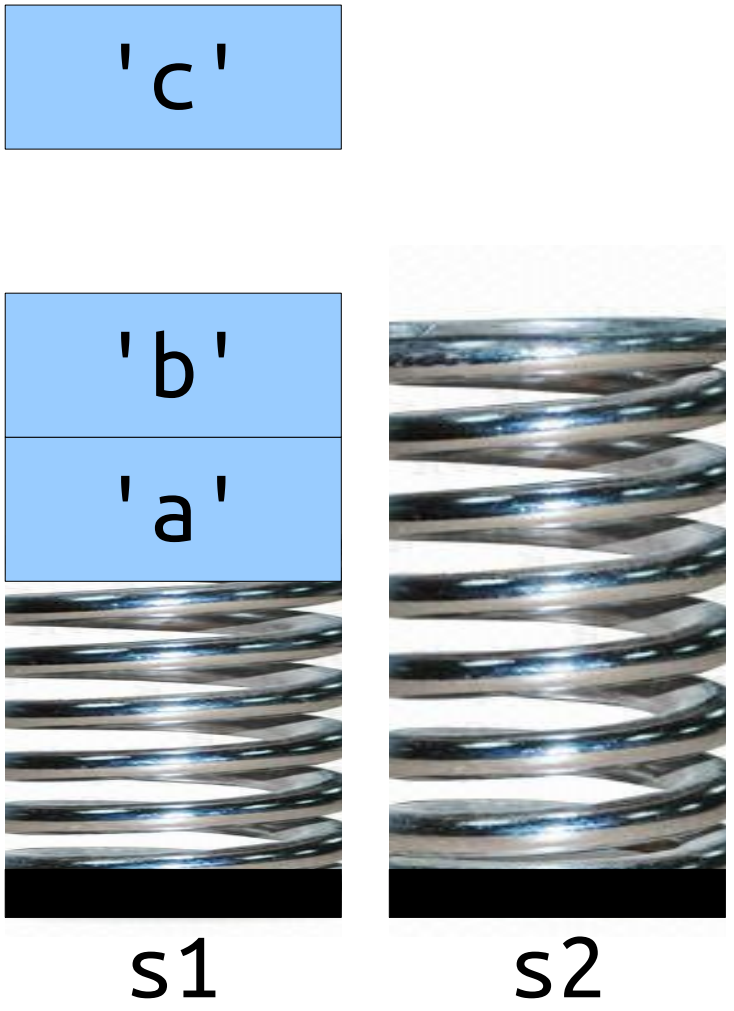
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

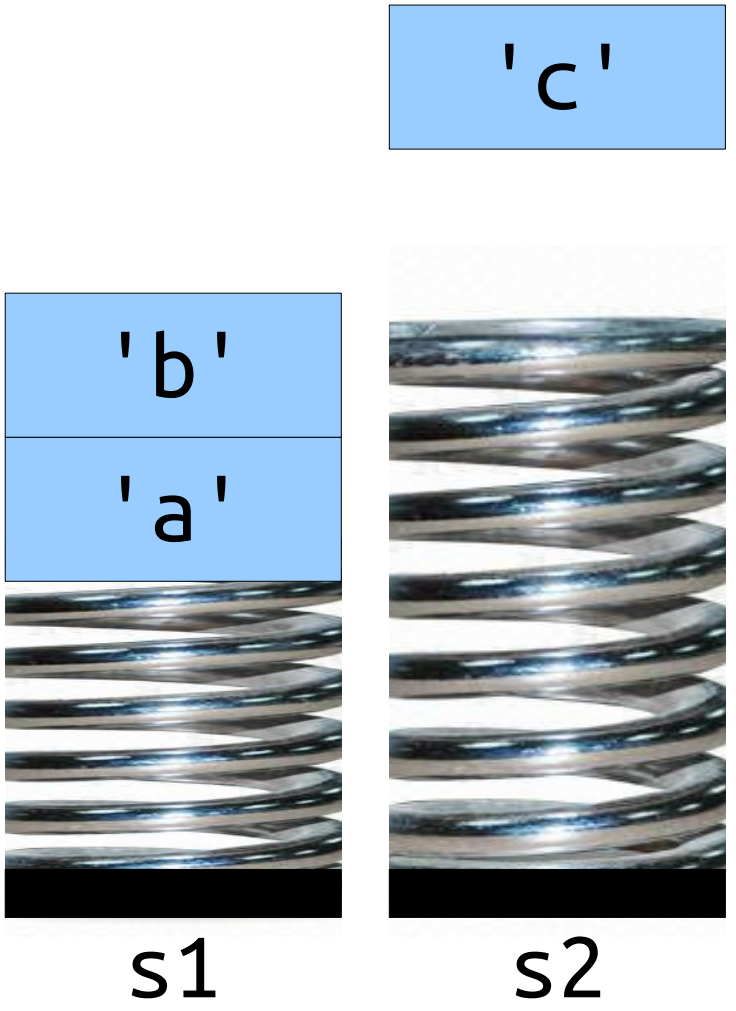
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

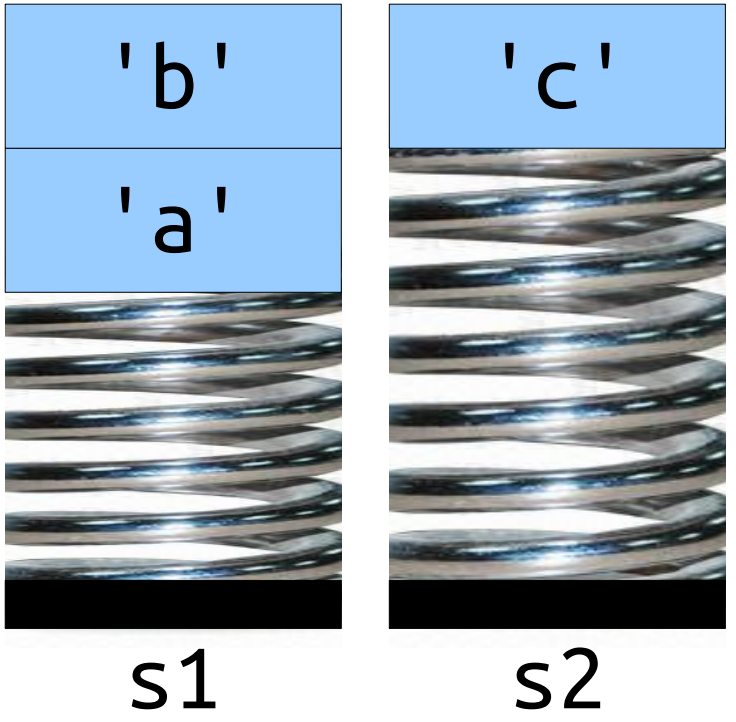
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

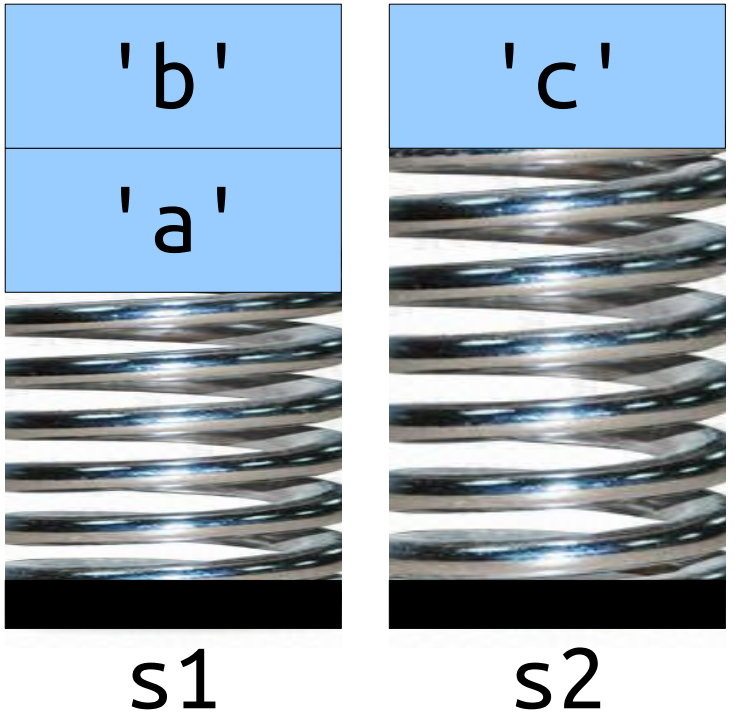
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

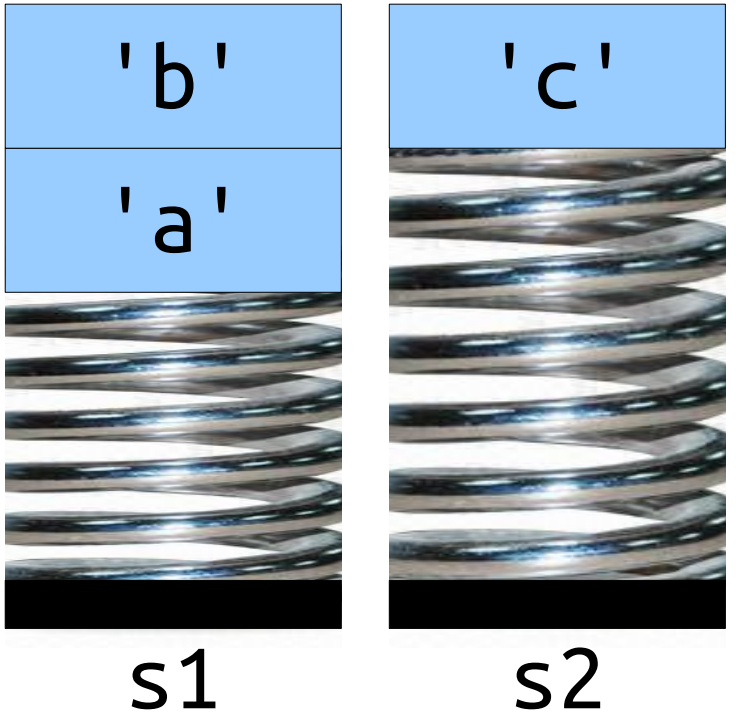
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

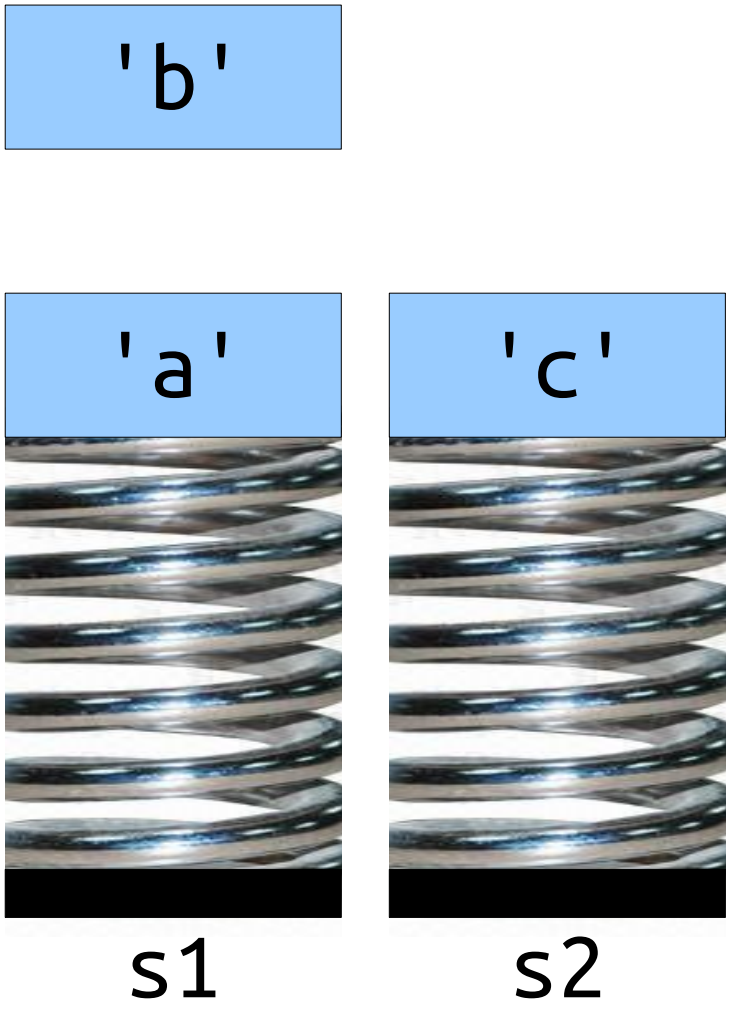
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

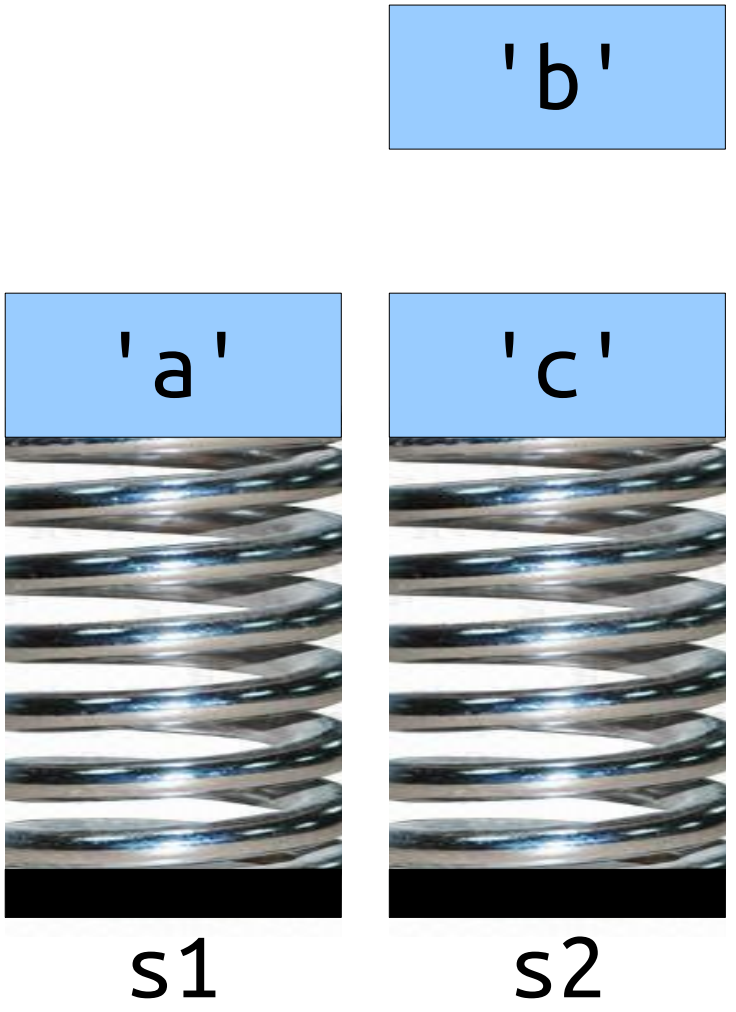
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

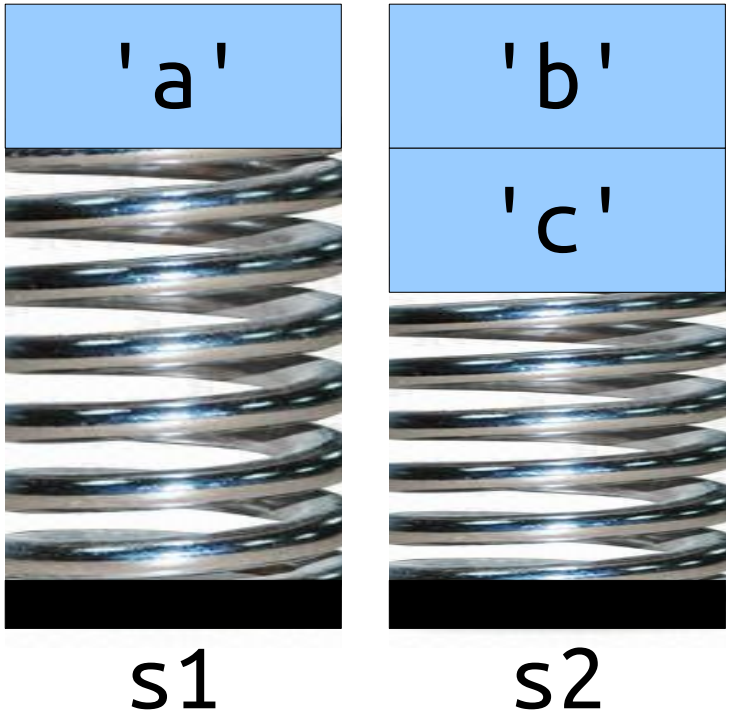
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

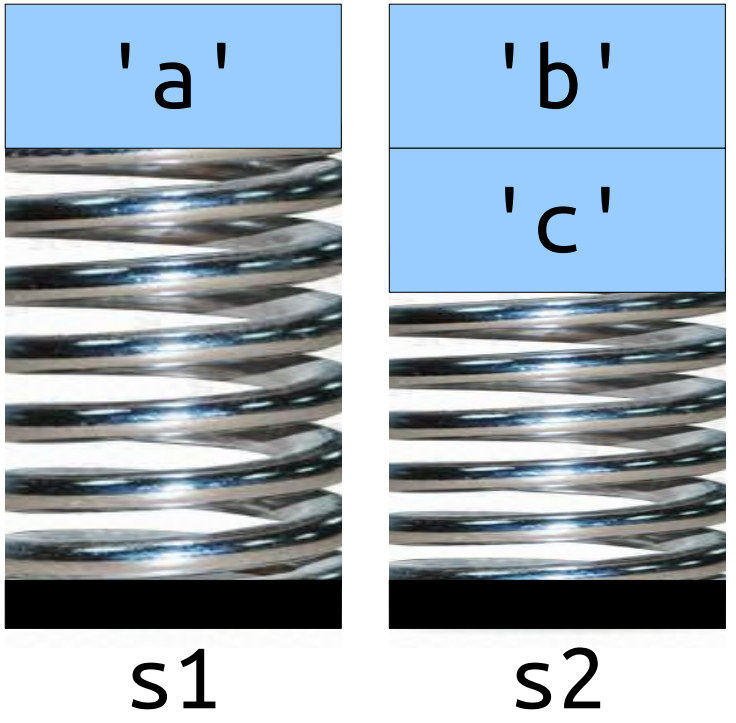
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

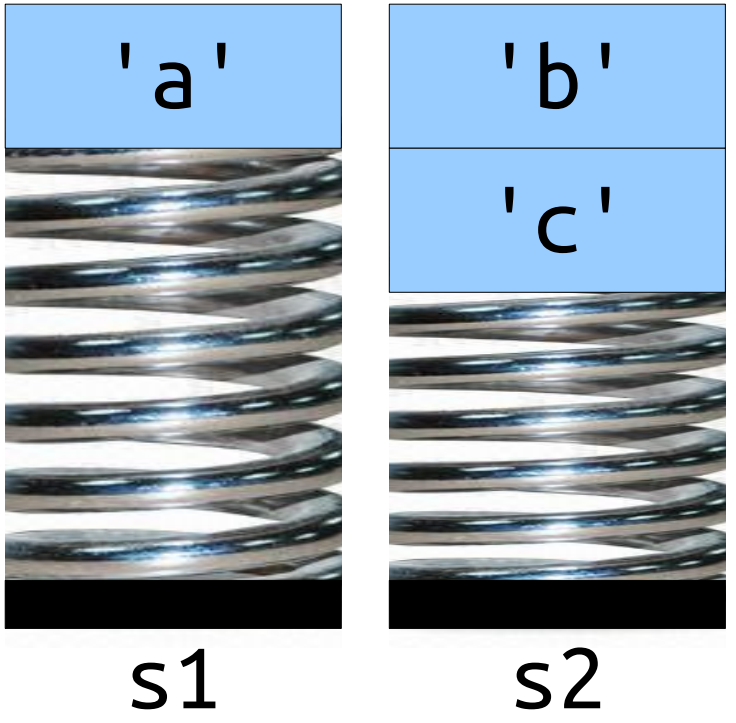
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

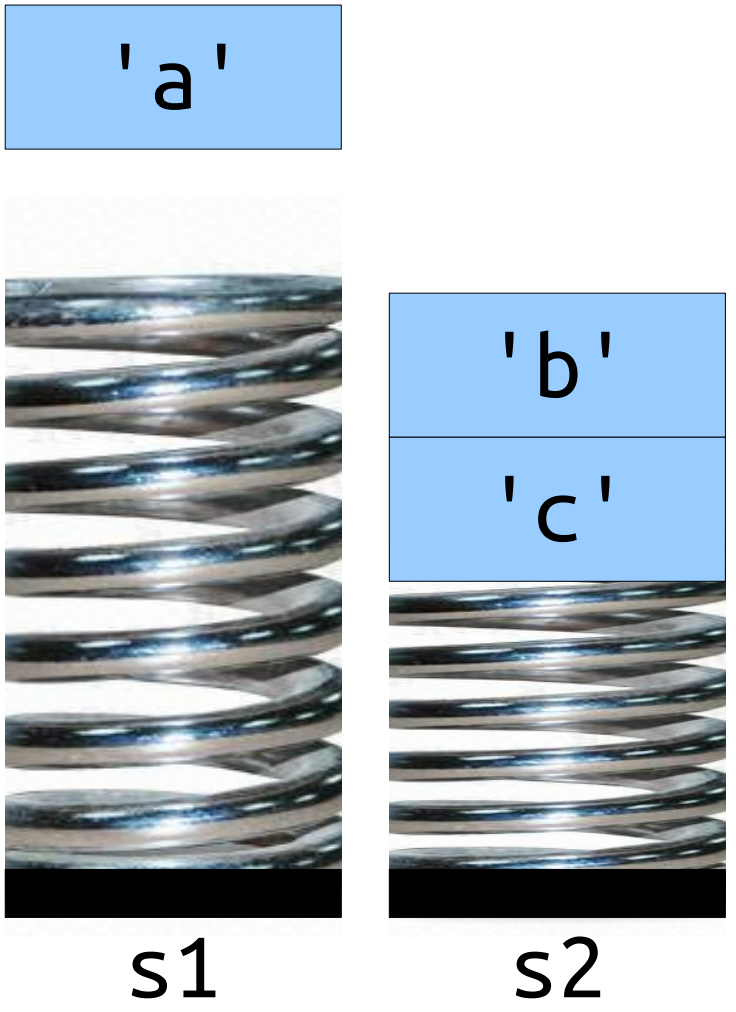
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

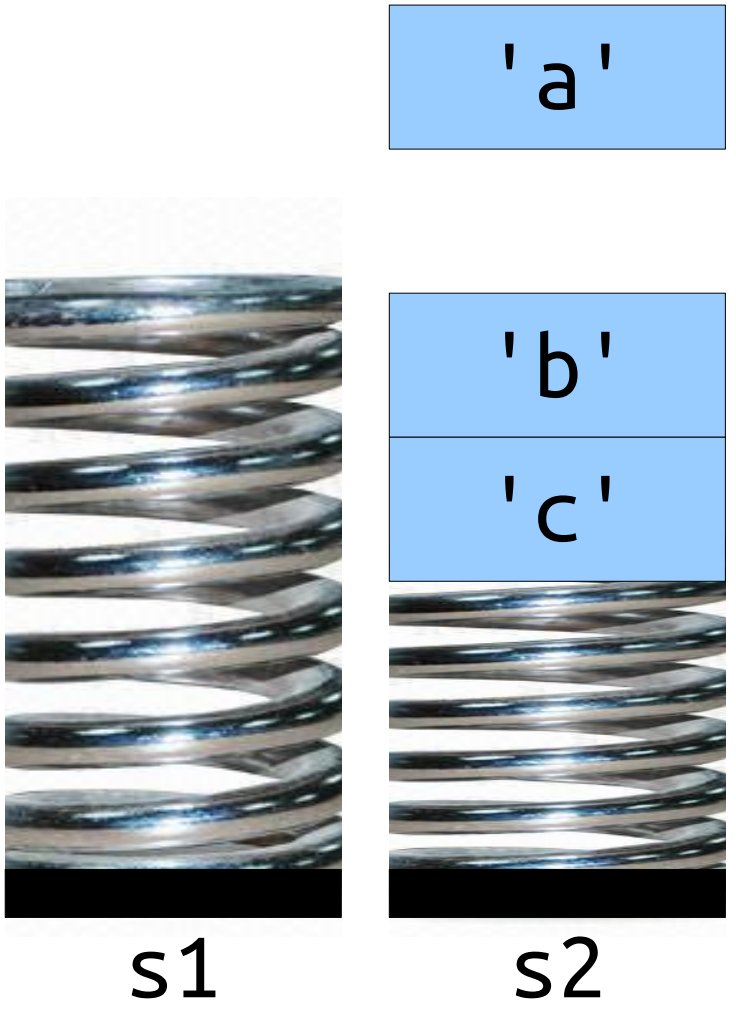
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

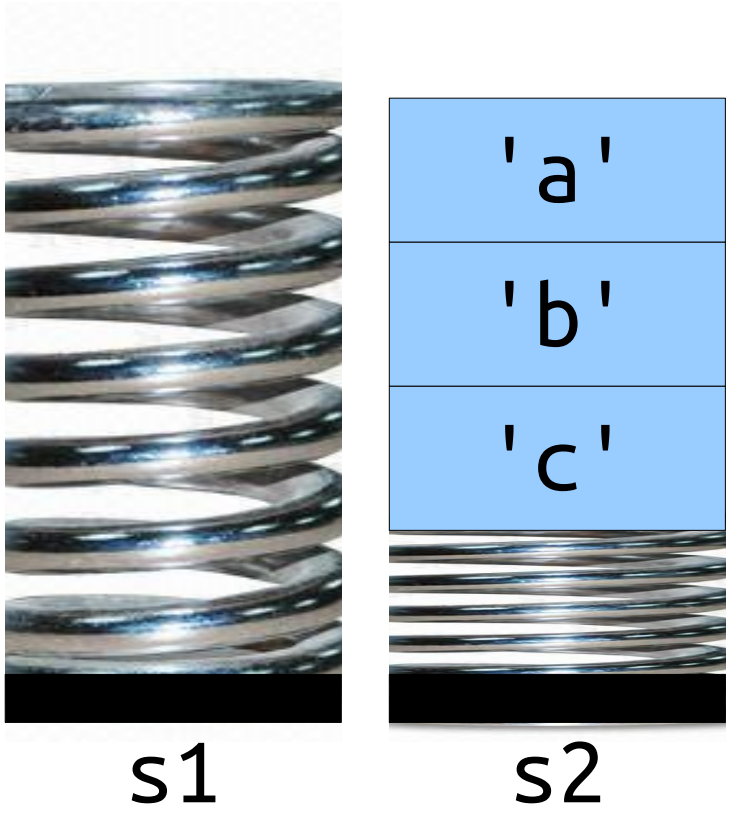
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

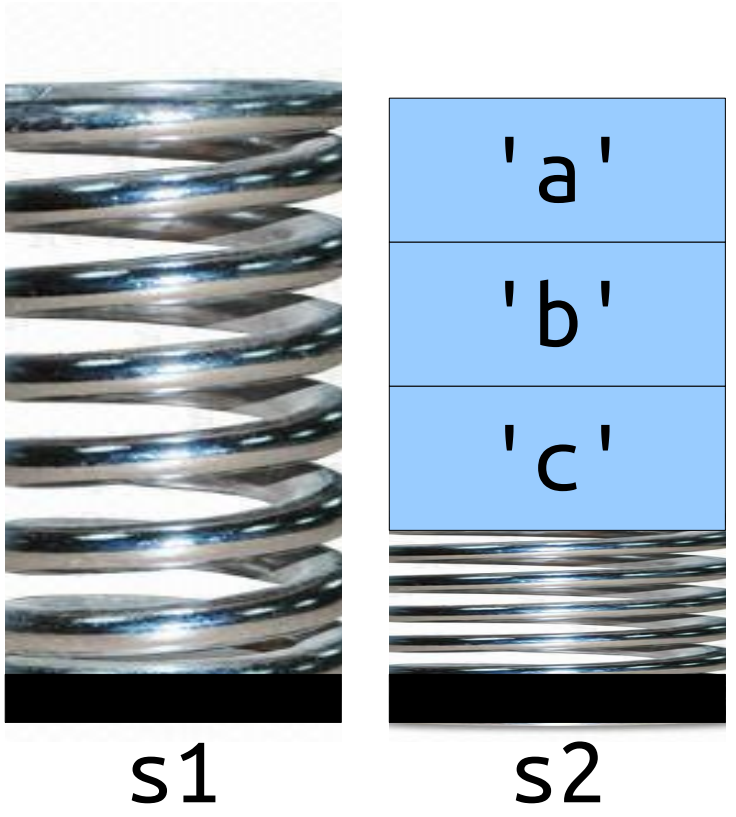
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

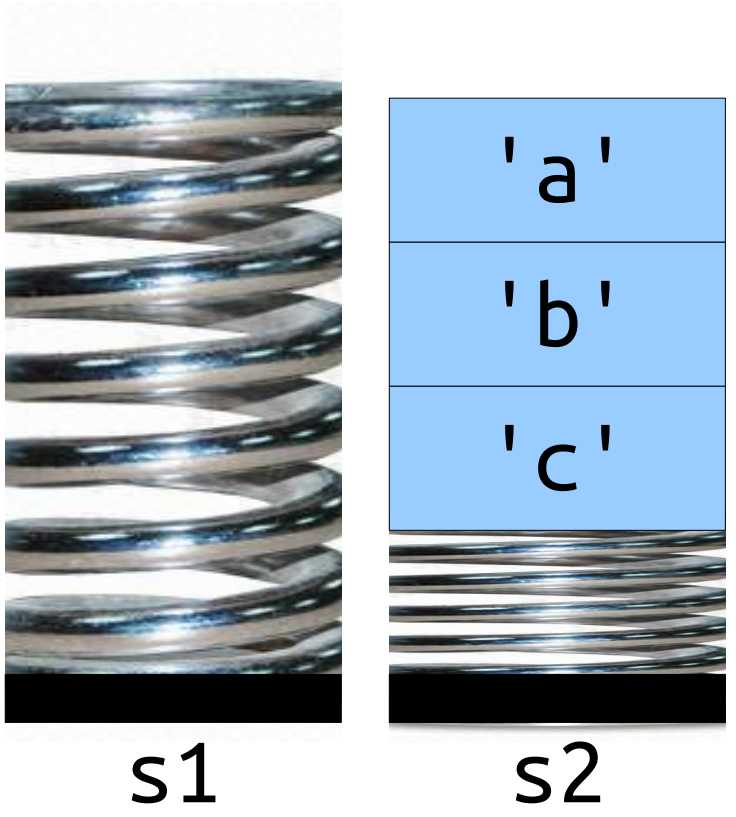
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

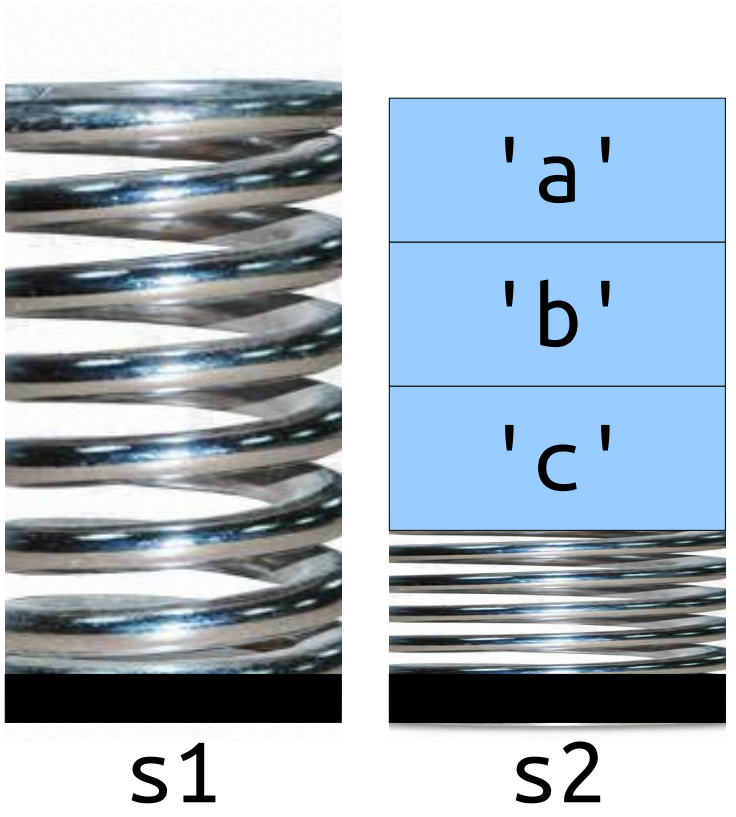
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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

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



Stack

What does this code print?

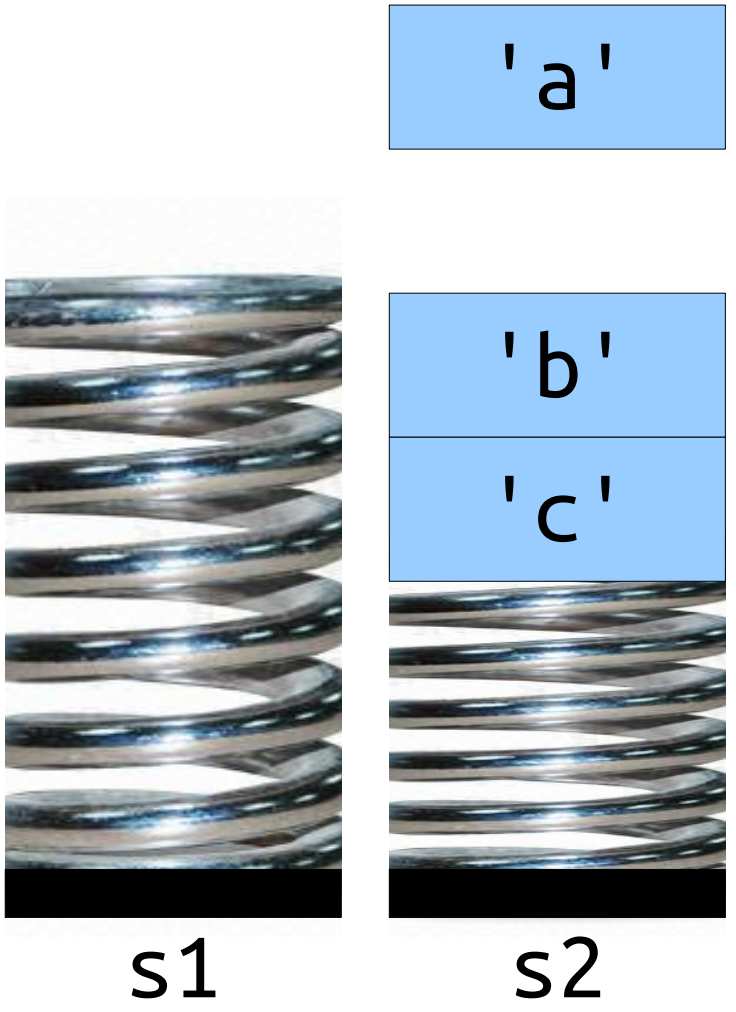
```
Stack<char> s1, s2;
s1.push('a');
s1.push('b');
s1.push('c');

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;
}
```

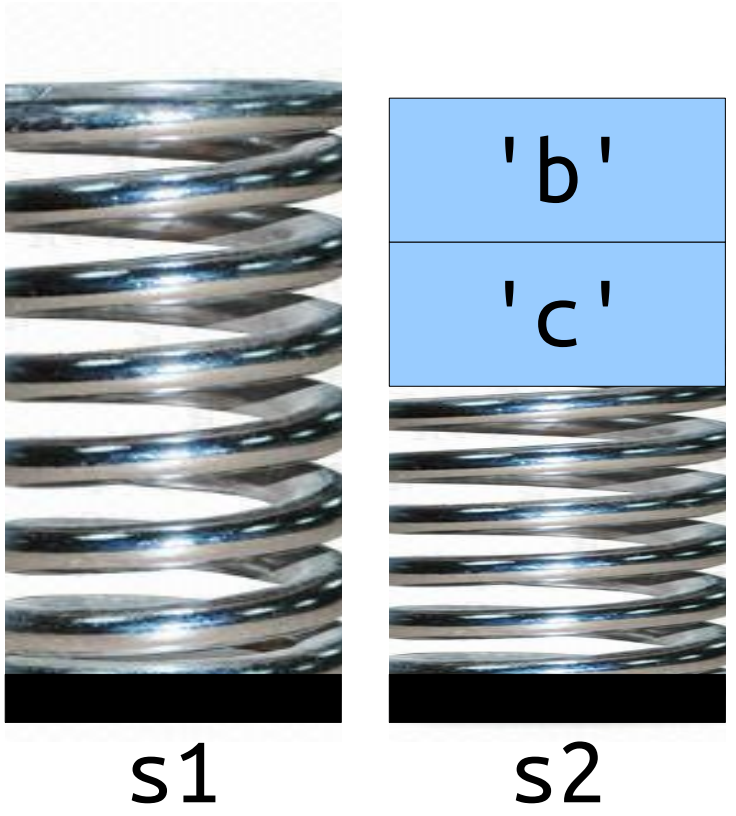


Stack

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'

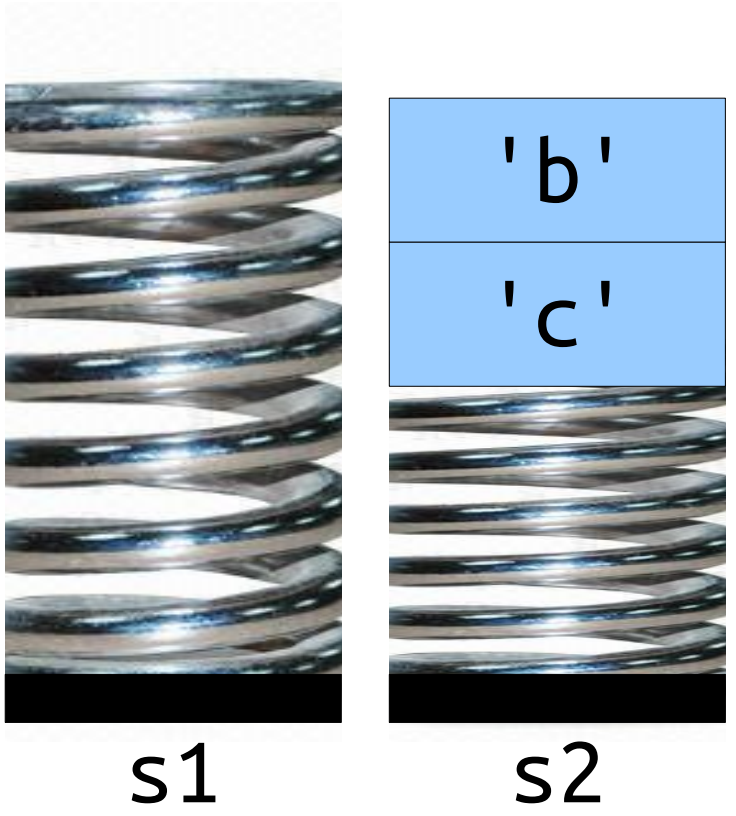


Stack

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'

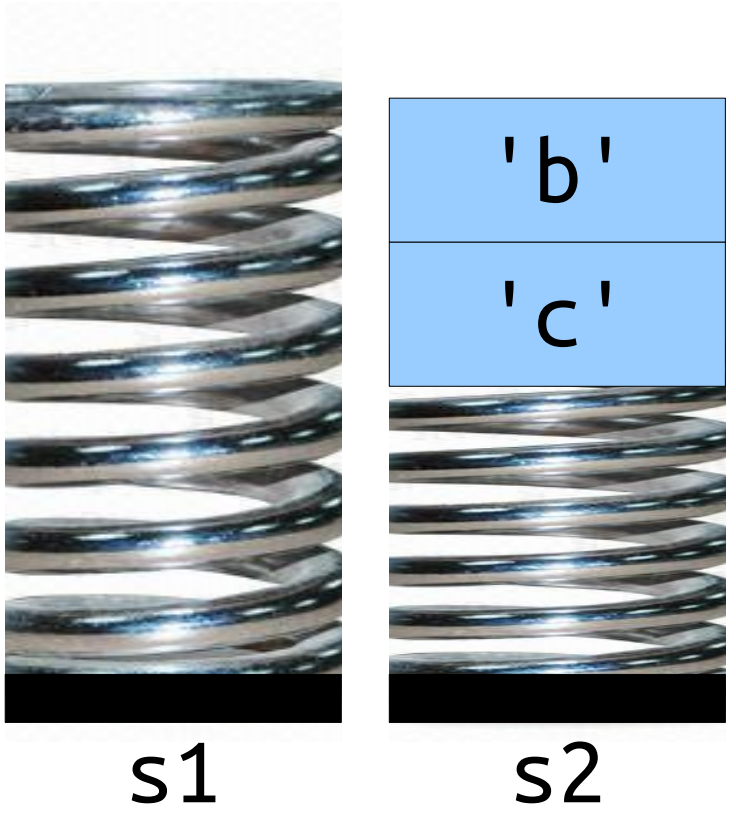


Stack

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'

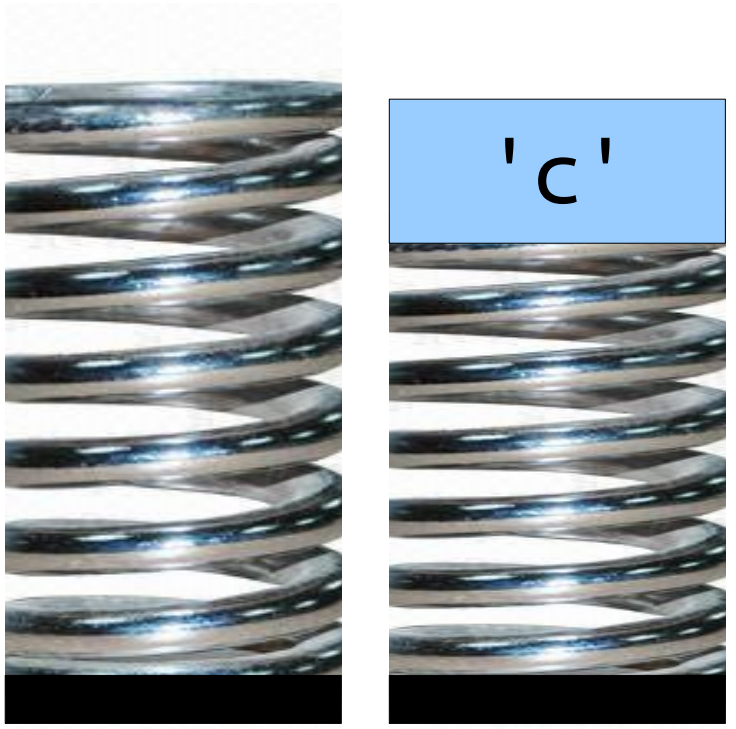


Stack

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;
}
```

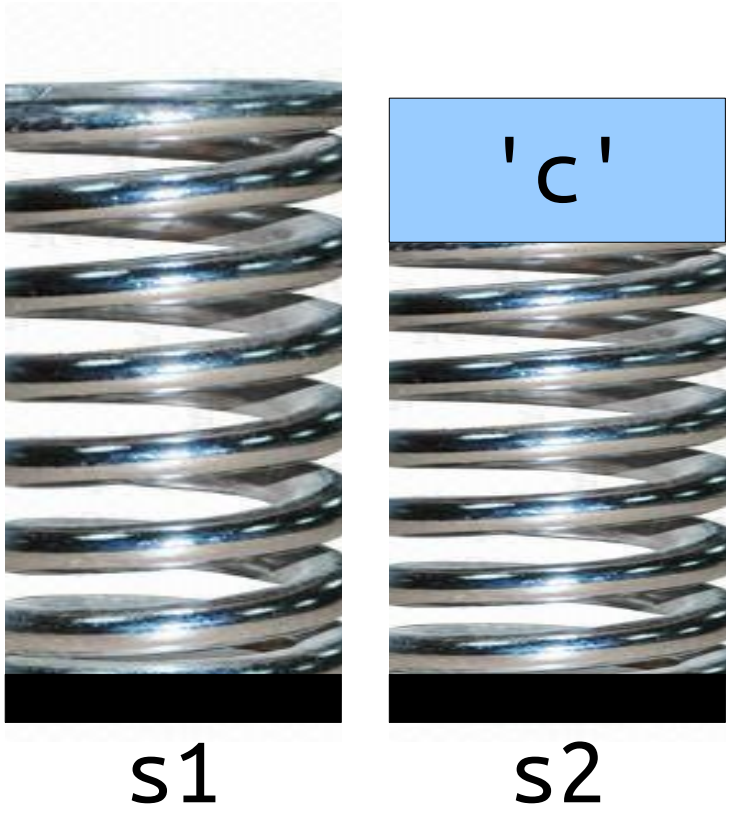
'a'



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;
}
```

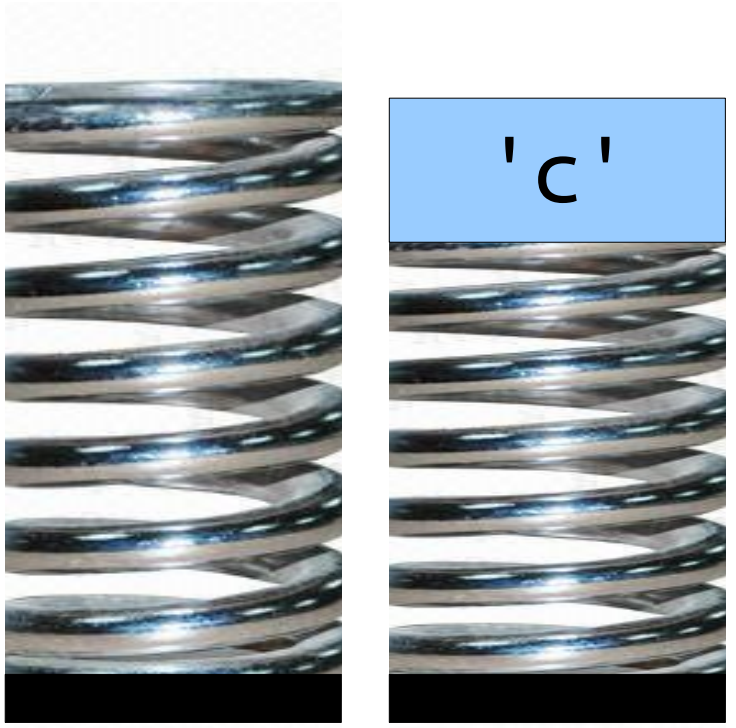


'a'	'b'

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;
}
```

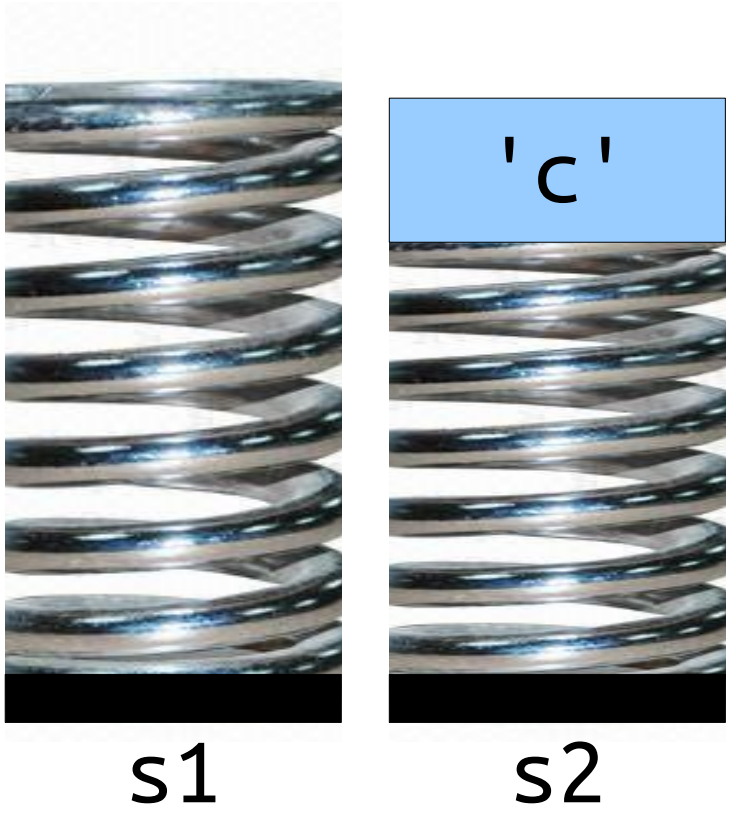


'a'	'b'

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;
}
```

s1 s2

'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;
}
```

s1 s2

'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;
}
```

s1 s2

'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

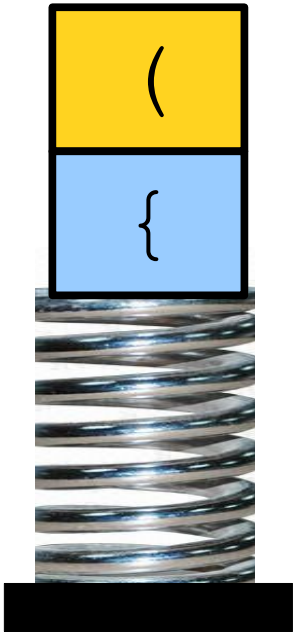
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
            ^
```

Balancing Parentheses

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

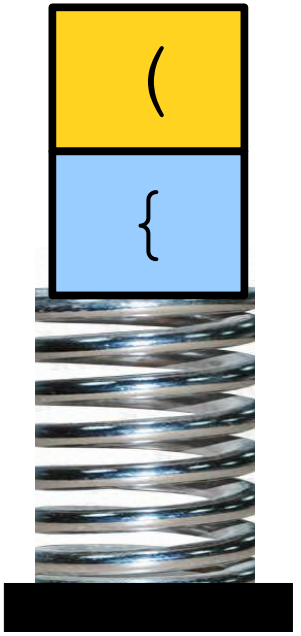
Balancing Parentheses

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



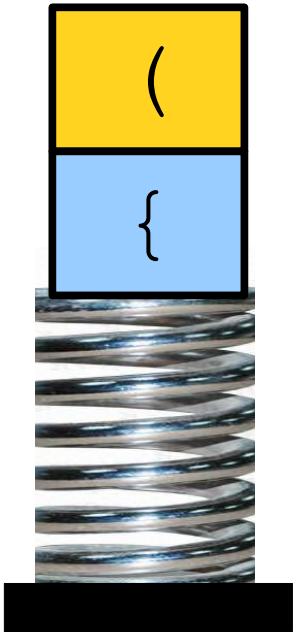
Balancing Parentheses

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



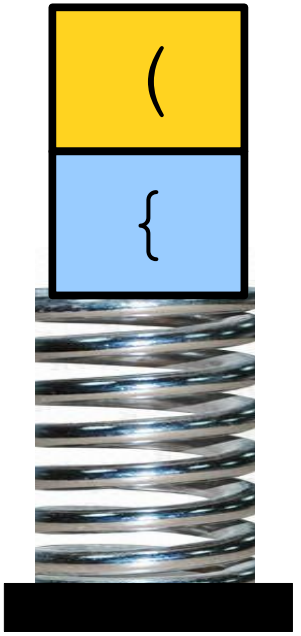
Balancing Parentheses

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



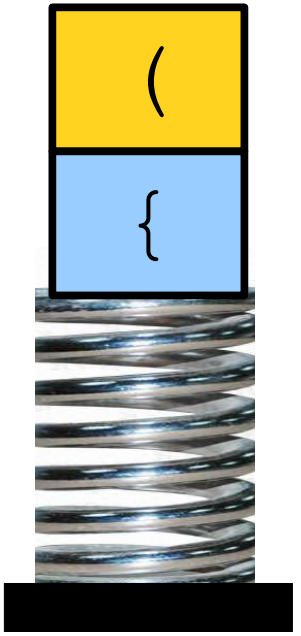
Balancing Parentheses

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



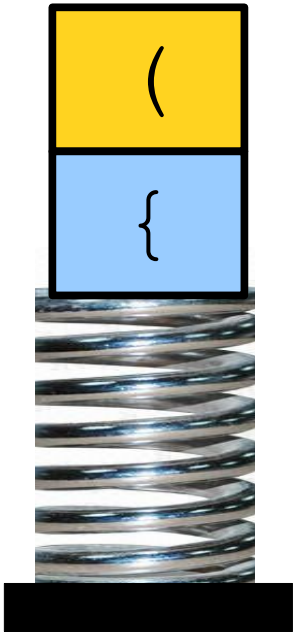
Balancing Parentheses

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



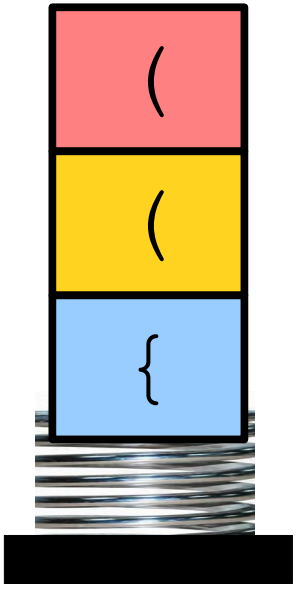
Balancing Parentheses

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



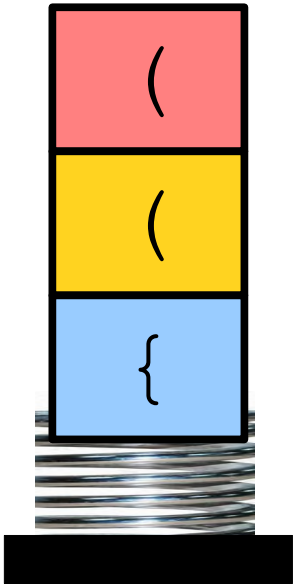
Balancing Parentheses

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



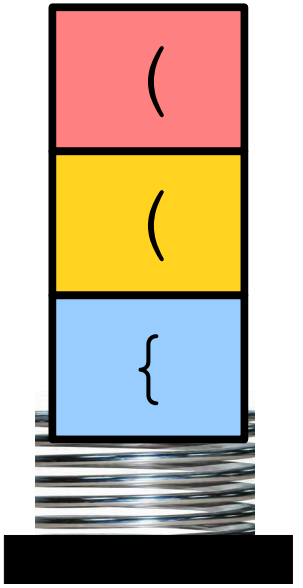
Balancing Parentheses

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



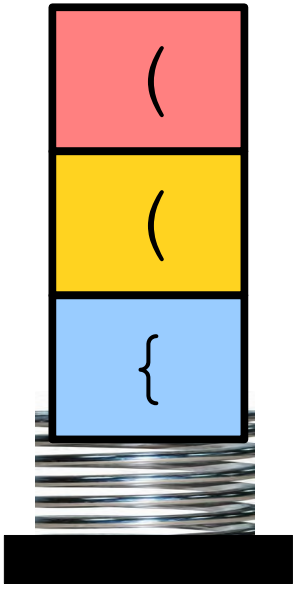
Balancing Parentheses

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



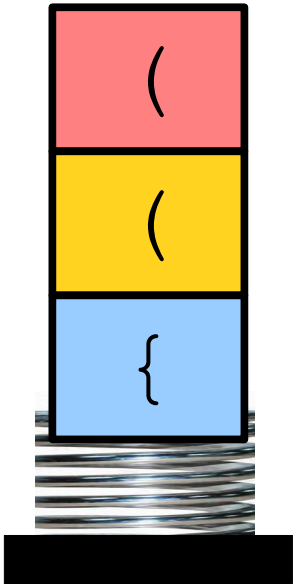
Balancing Parentheses

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



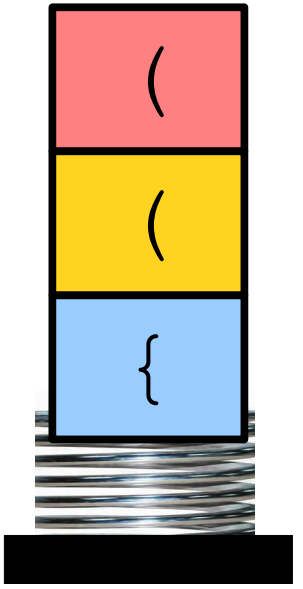
Balancing Parentheses

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



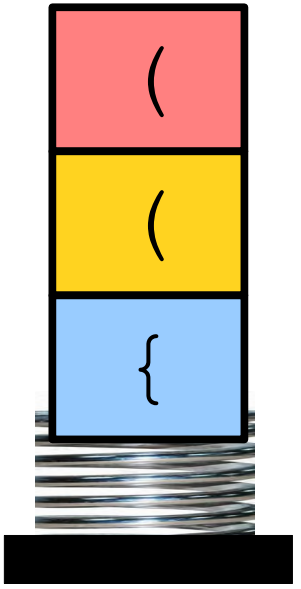
Balancing Parentheses

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



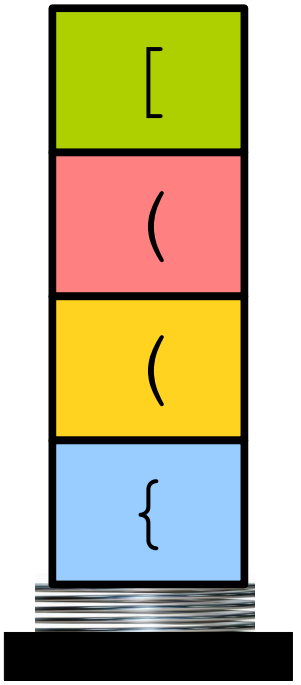
Balancing Parentheses

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



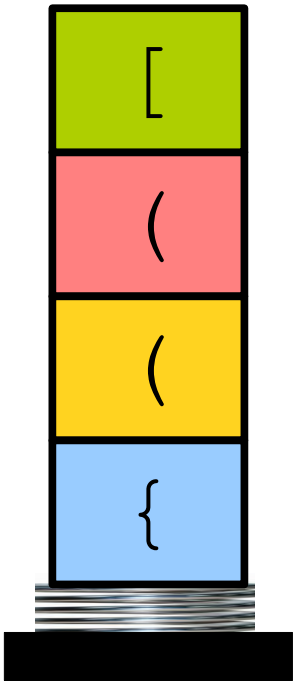
Balancing Parentheses

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



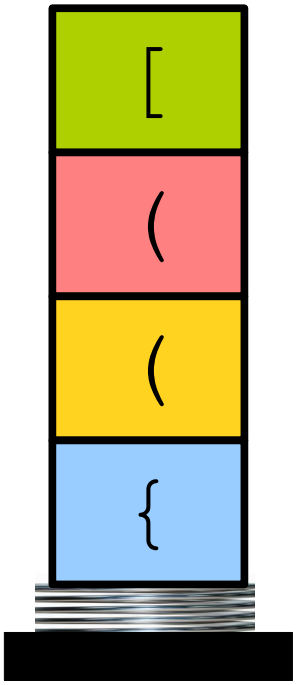
Balancing Parentheses

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



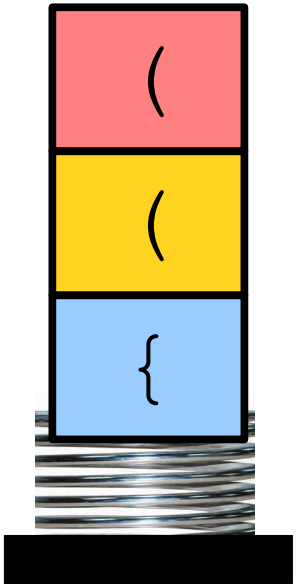
Balancing Parentheses

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



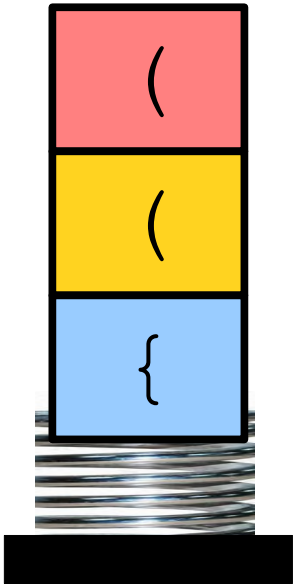
Balancing Parentheses

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



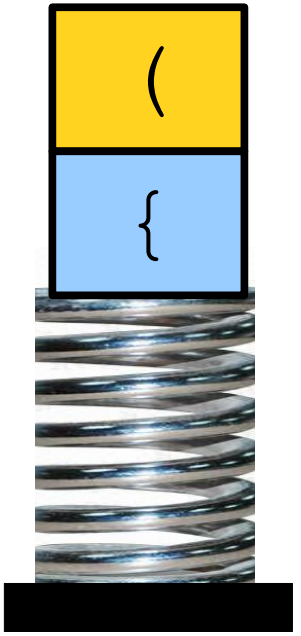
Balancing Parentheses

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



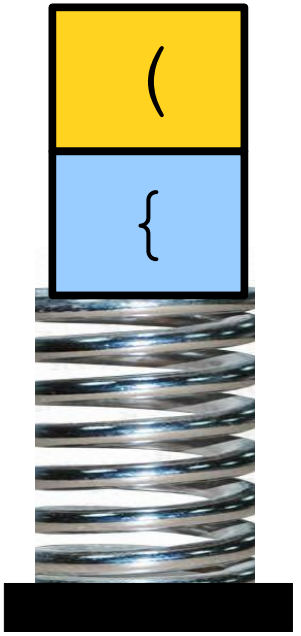
Balancing Parentheses

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



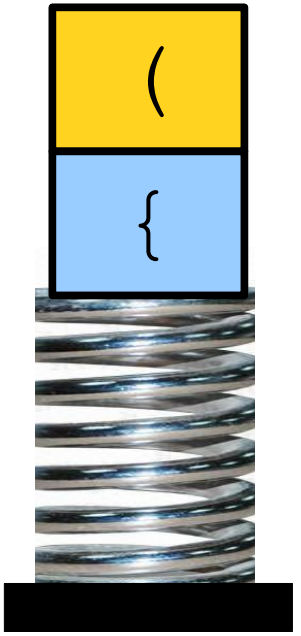
Balancing Parentheses

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



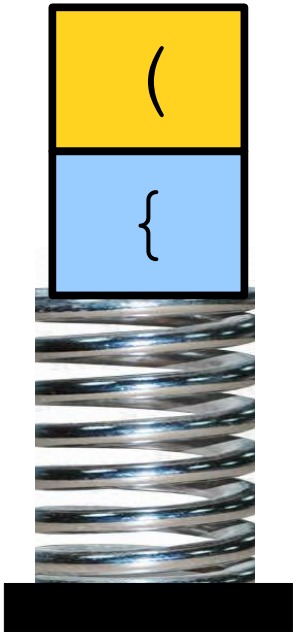
Balancing Parentheses

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



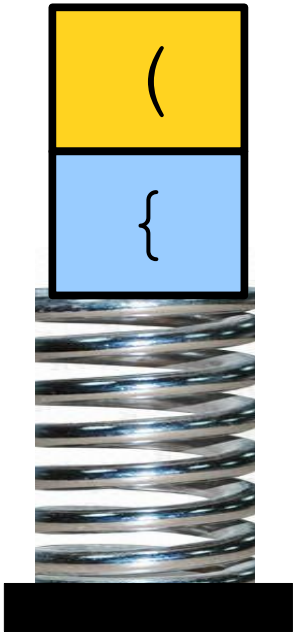
Balancing Parentheses

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



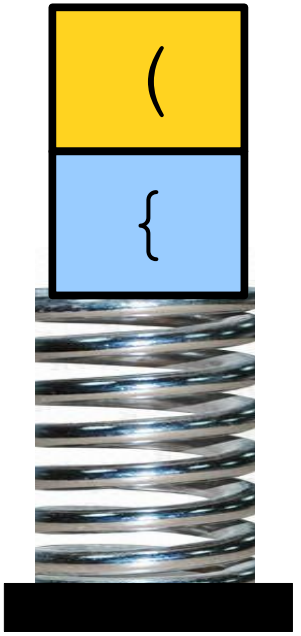
Balancing Parentheses

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



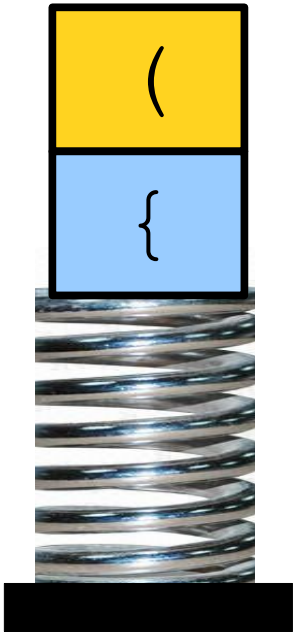
Balancing Parentheses

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



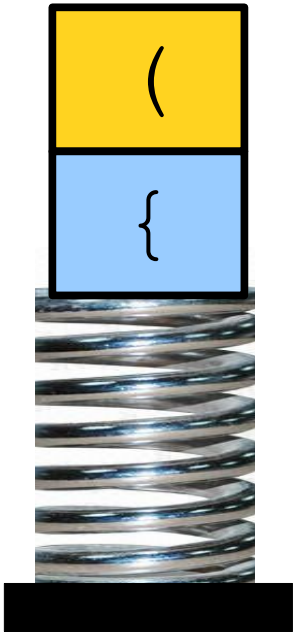
Balancing Parentheses

```
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

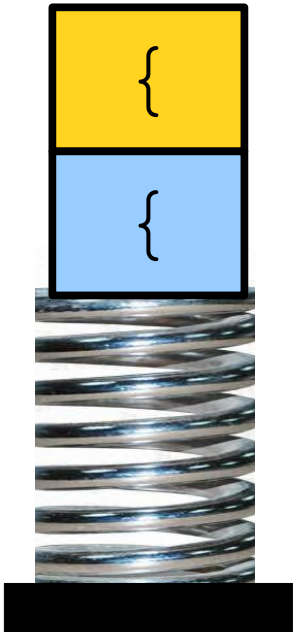
```
int foo() { if (x * (y + z[1]) < 137) { x = 1; } }
                                      ^
```

Balancing Parentheses

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

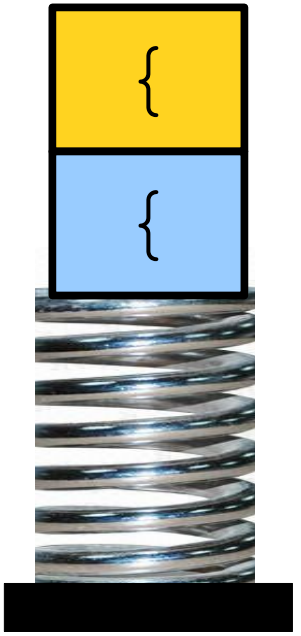
Balancing Parentheses

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



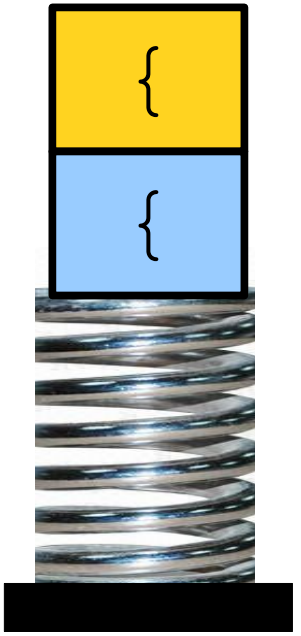
Balancing Parentheses

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



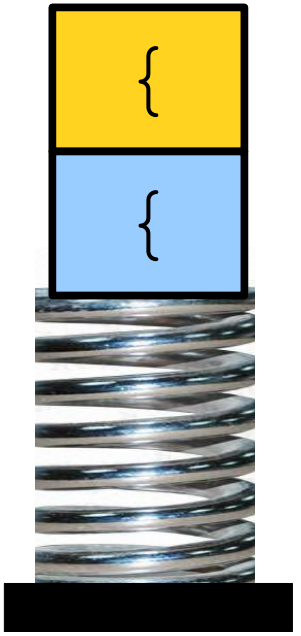
Balancing Parentheses

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



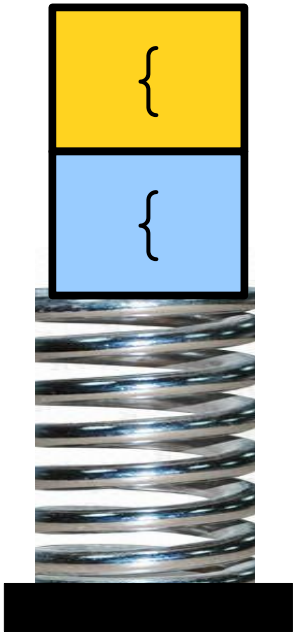
Balancing Parentheses

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



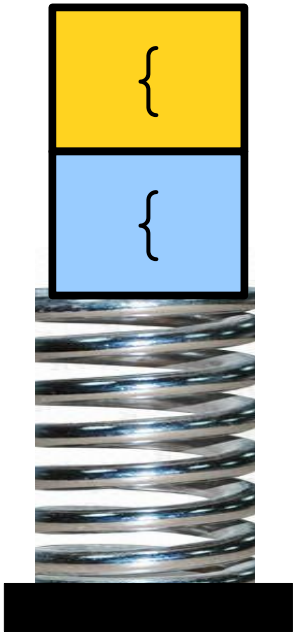
Balancing Parentheses

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



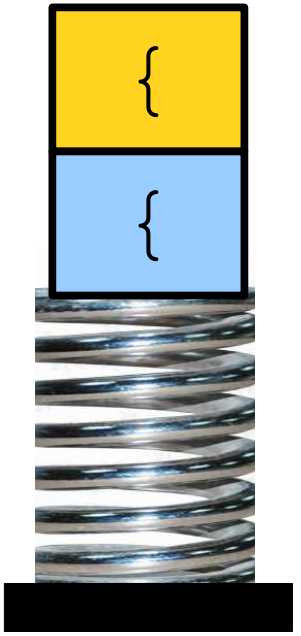
Balancing Parentheses

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



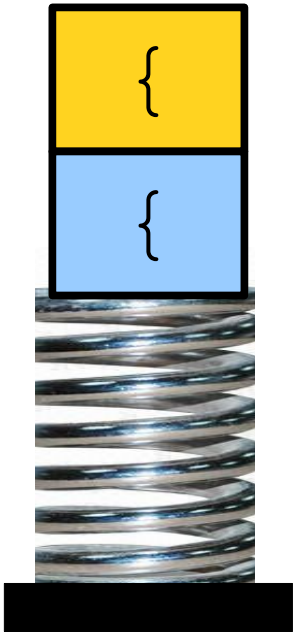
Balancing Parentheses

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



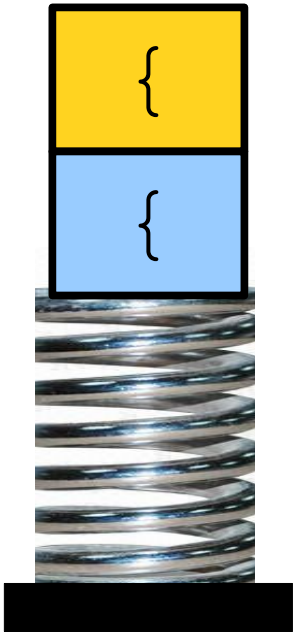
Balancing Parentheses

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



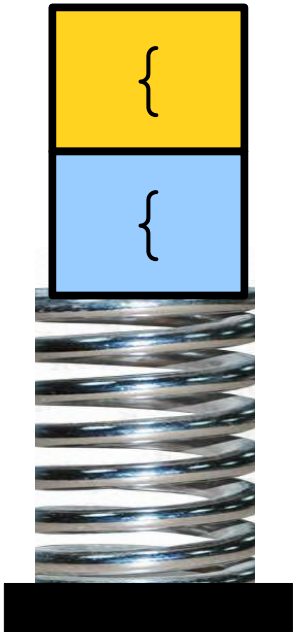
Balancing Parentheses

```
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

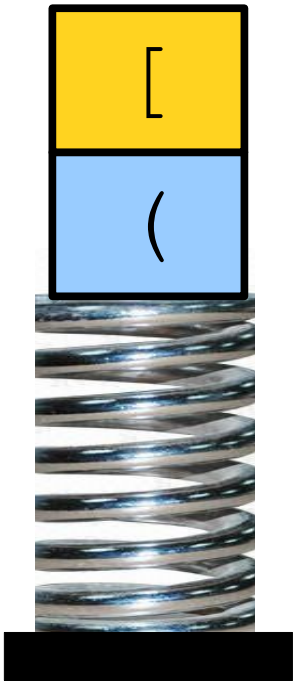
```
( [ ) ]
^
```

Balancing Parentheses

```
( [ ) ]
  ^
```

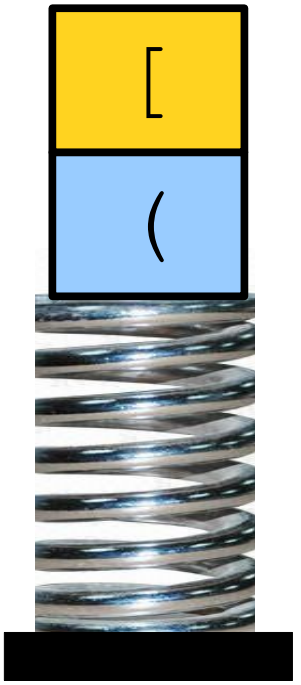
Balancing Parentheses

```
( [ ) ]
  ^
```

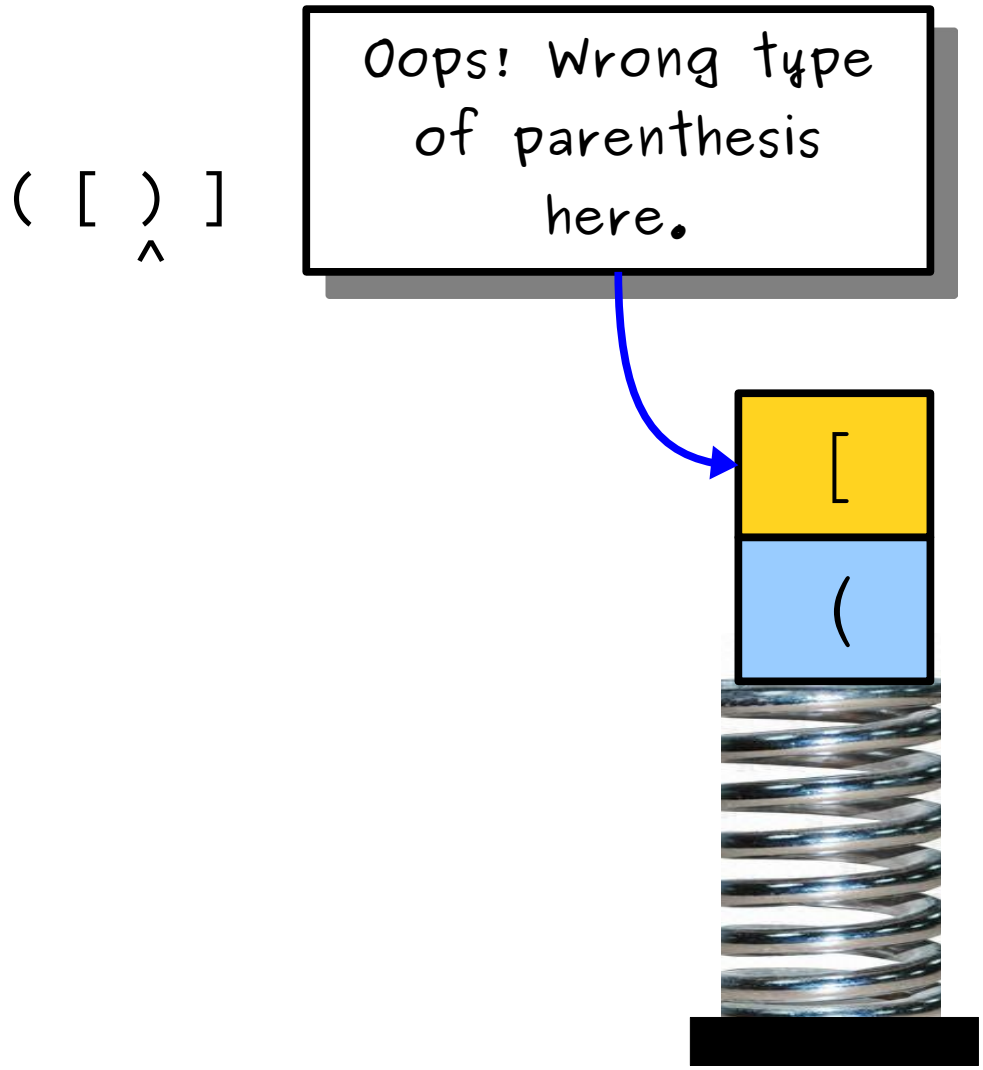


Balancing Parentheses

```
( [ ) ]
    ^
```



Balancing Parentheses



Balancing Parentheses

((

Balancing Parentheses

((
^

Balancing Parentheses

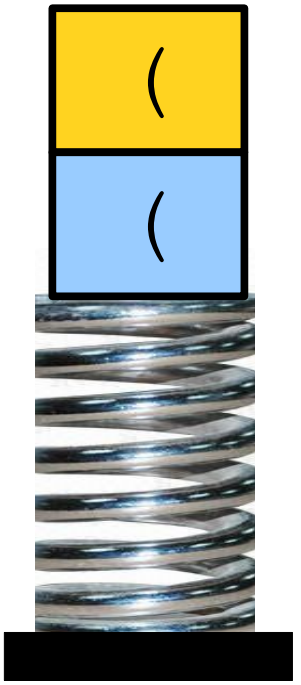
((
^

Balancing Parentheses

((
  ^

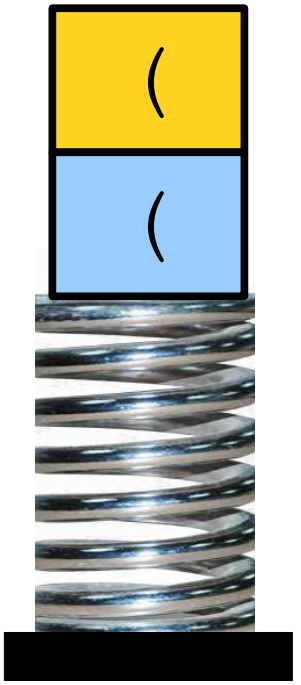
Balancing Parentheses

((
  ^



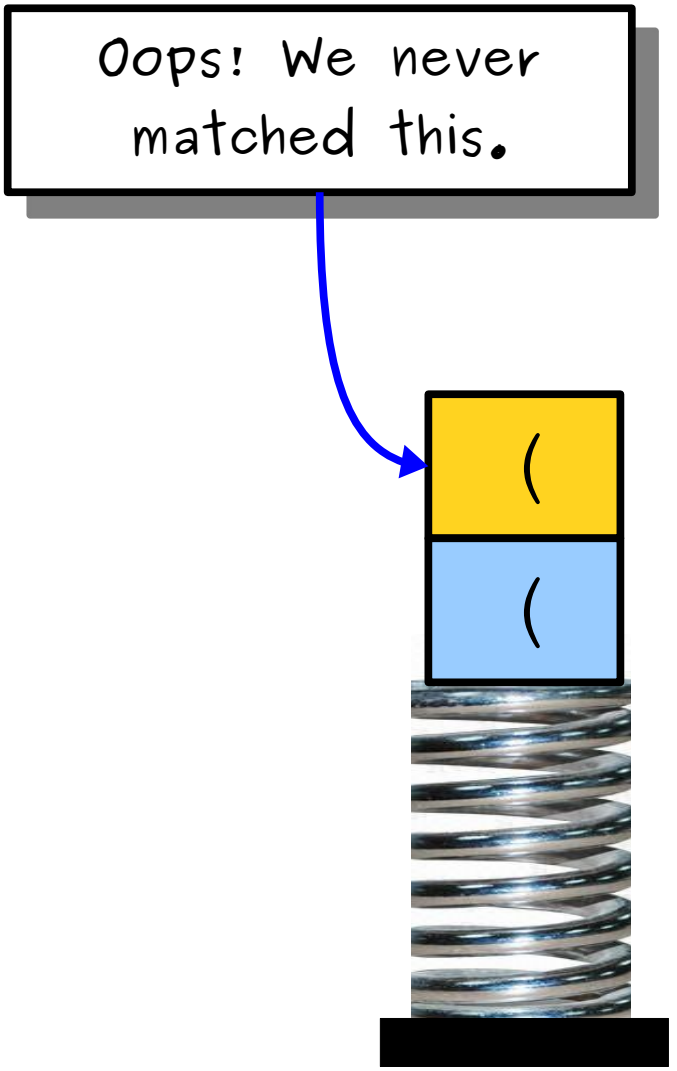
Balancing Parentheses

((



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!

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.
- If you’d like to discuss ethics in technology more, feel free to stop by or call into my “Chat About Anything” hours today from 3PM – 5PM in Durand 317.

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.

School of Engineering
Department of Computer Science

Info Session #1:

CS Major & CS Minor Declarations

Submit Questions Here

January 24, 2023
4PM - 5PM
Zoom Info:
https://tinyurl.com/25eda4x6

Upcoming Career Fair

The Computer Forum Career Fair will be held Wednesday, January 25, in-person. Sign-ups are now open! Stanford students only; student IDs required at check-in.

Date: Wednesday, January 25
Time: 11:00am - 4:00pm
Location: Arrillaga Center for Sports and Recreation, Basketball Courts [Enter through the doors that lead directly to the courts, not through the main ACSR entrance]

Register via Handshake. Career Fair Plus link.

We will do in-person scheduled only sessions for the first three hours (11am - 2pm) and open to all for the last two hours (2pm - 4pm). There is an enforced 10 minute buffer between each session you are able to sign up for to allow for you to find the company/recruiter.

Candidate Checklist, How to Book Meetings.

Best Practices:

- Be courteous of employers and other students. Only sign up for what you will attend.
- Only sign up for one session per company
- Continuously check back to see newly added schedules.

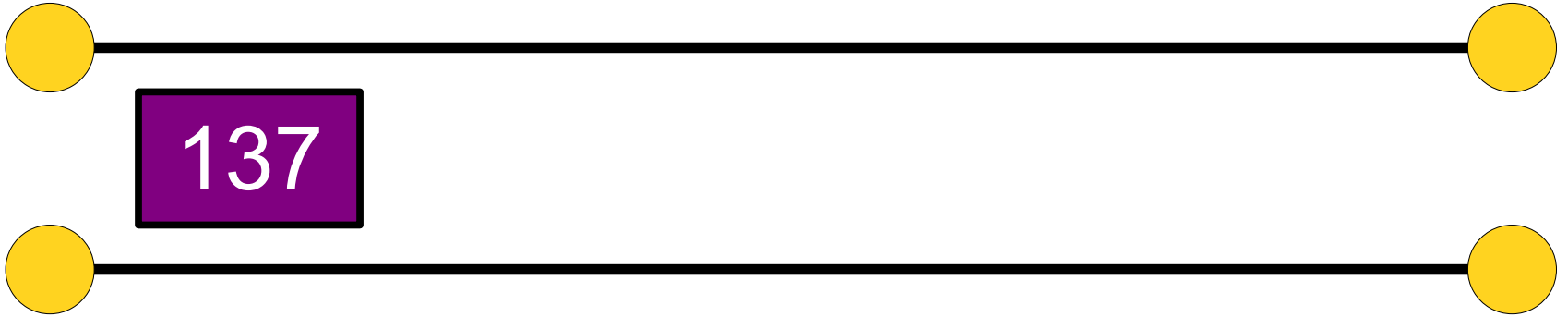
Stanford Computer Forum policies regarding no-shows: if you do not show up for a session you signed up for, your participation in future Computer Forum events may be revoked. By signing up and not showing up, you are taking away a spot from another student.

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

Queue

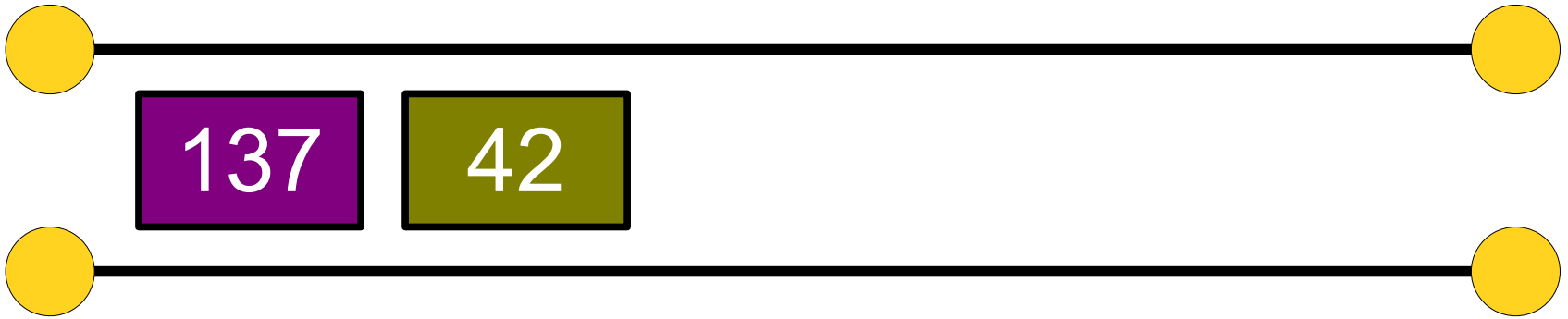
Queue

- 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.



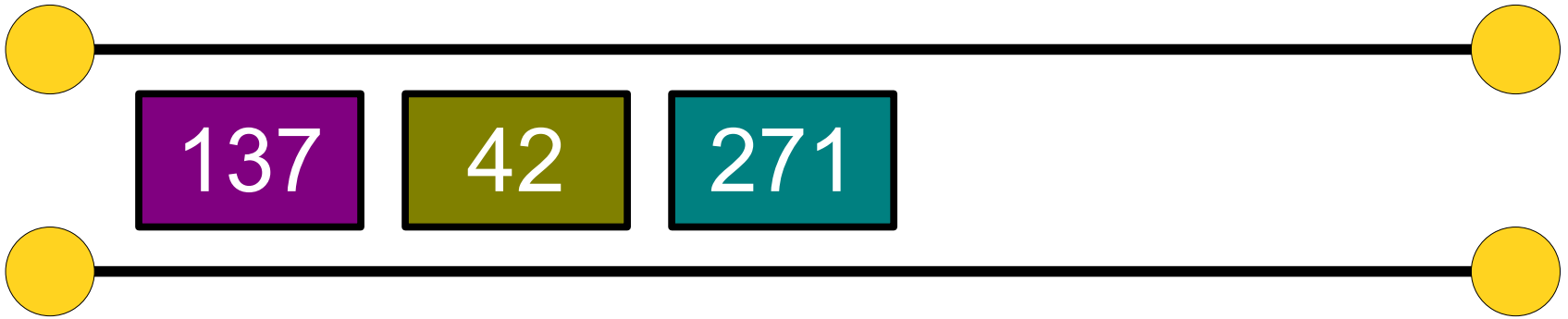
Queue

- 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.



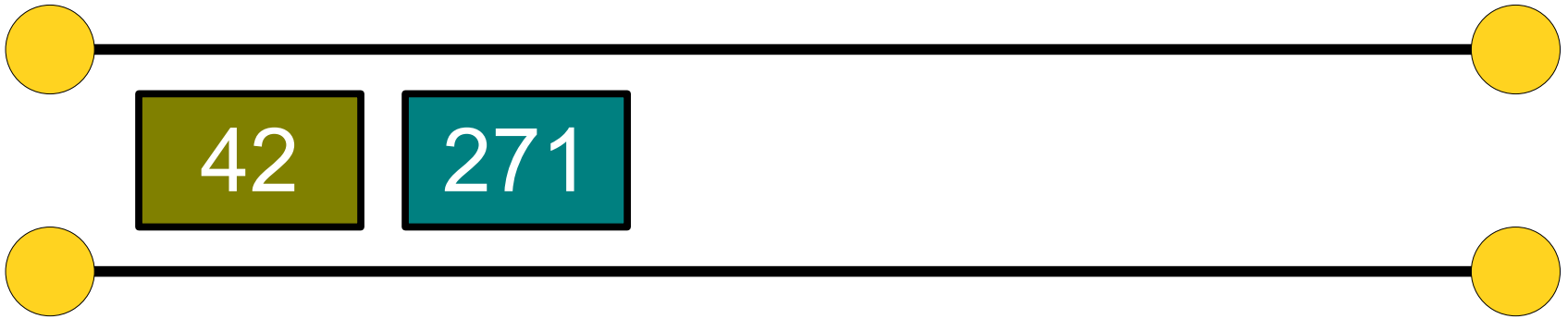
Queue

- 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.



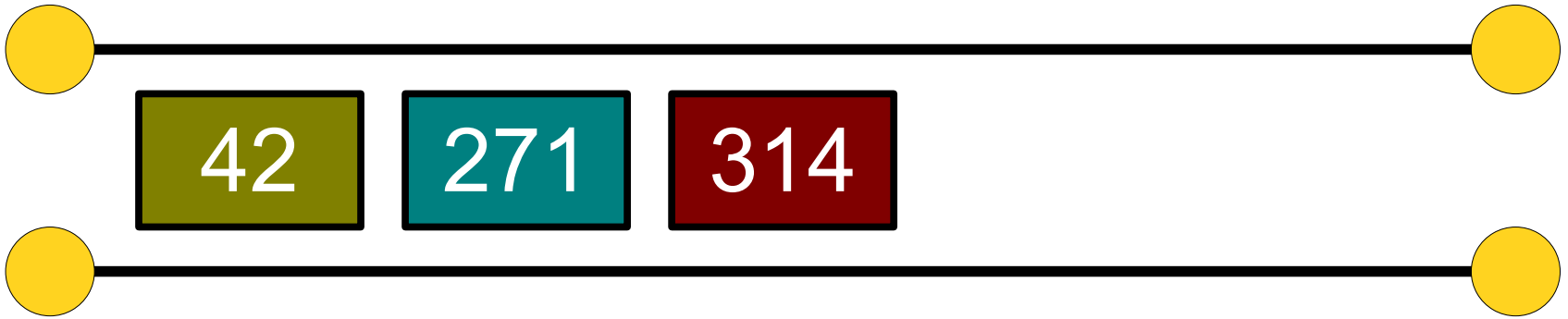
Queue

- 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.



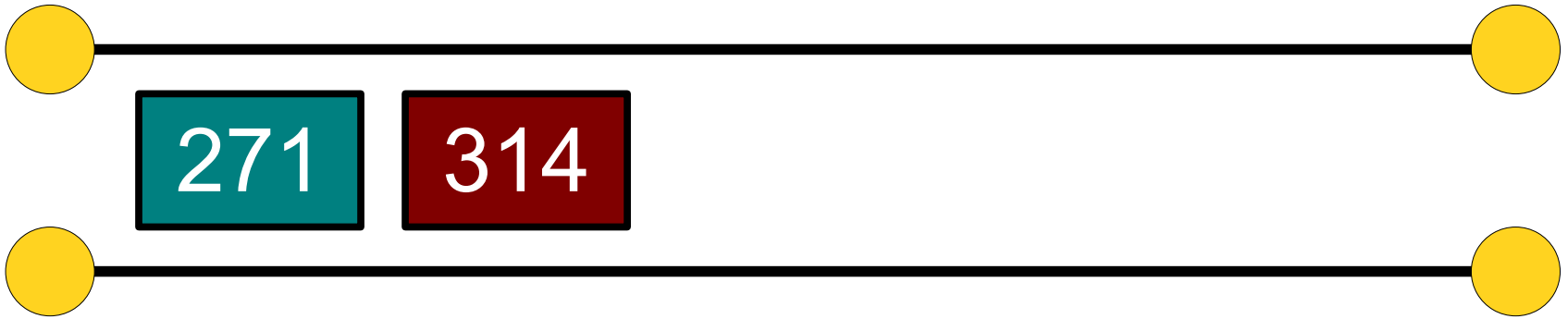
Queue

- 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.



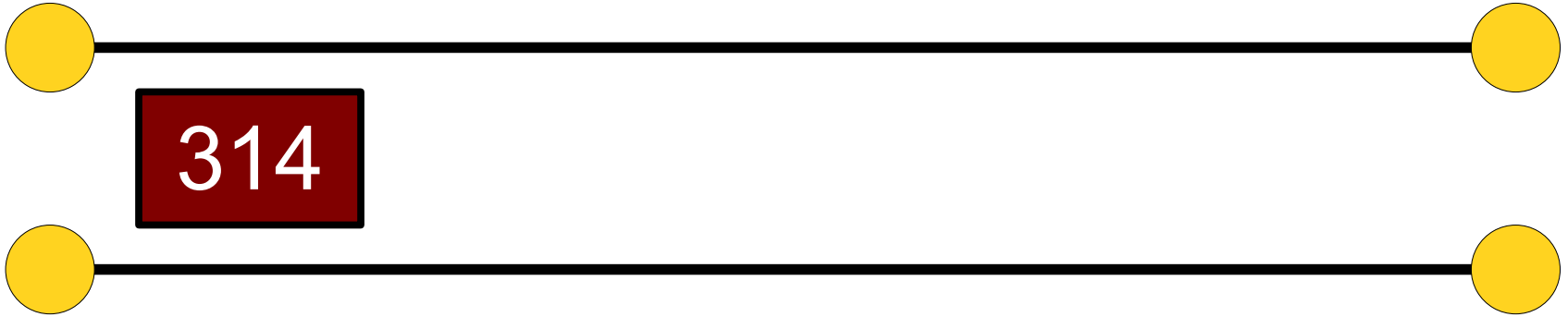
Queue

- 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?

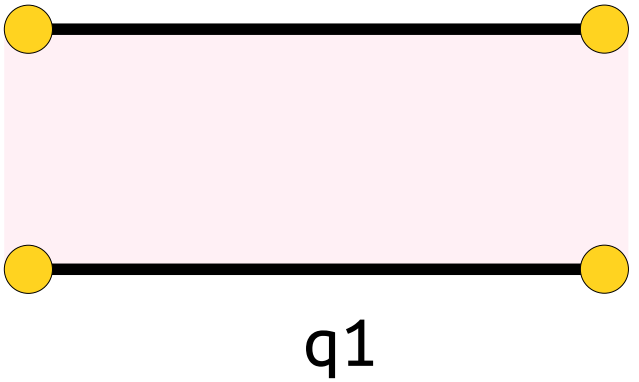
```
Queue<char> q1, q2;
q1.enqueue('a');
q1.enqueue('b');
q1.enqueue('c');

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

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

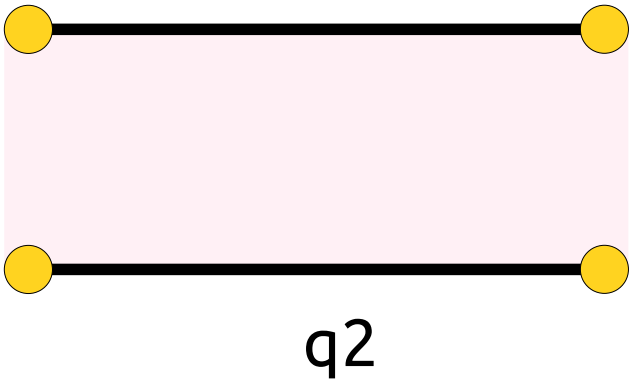
Answer at
https://pollev.com/cs106bwin23

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;
}
```

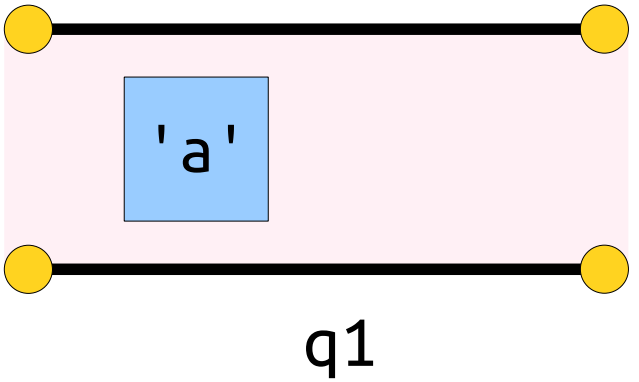


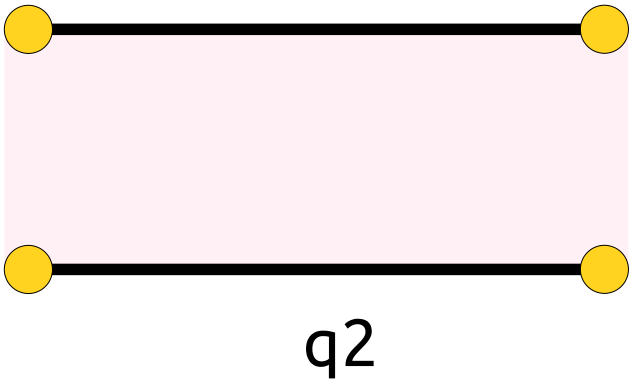


Queue

- 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;
}
```

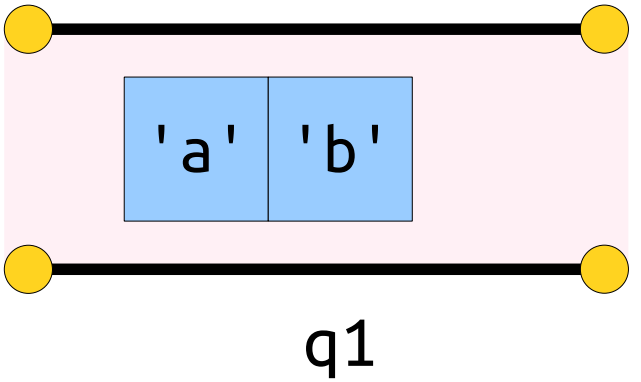


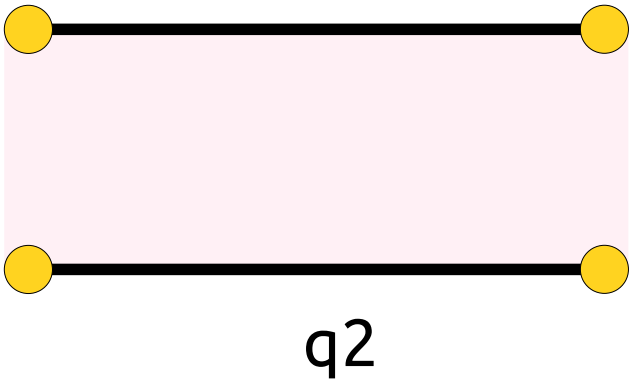


Queue

- 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;
}
```

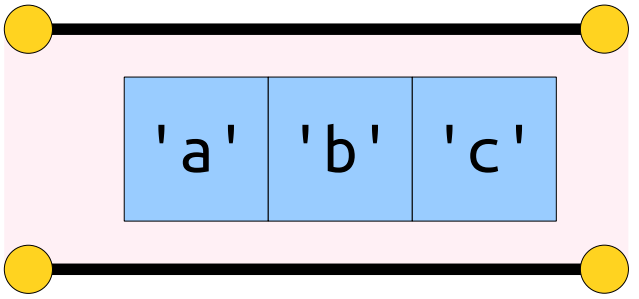




Queue

- 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;
}
```



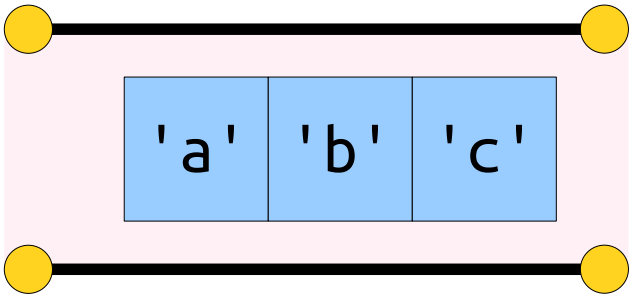
q1

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;
}
```



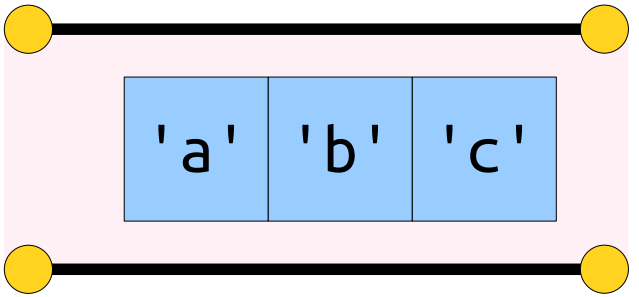
q1

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;
}
```



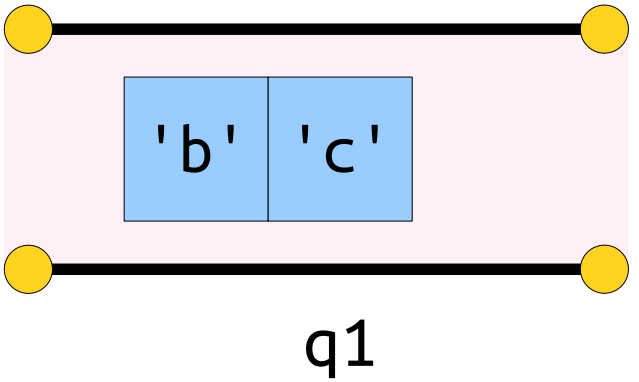
q1

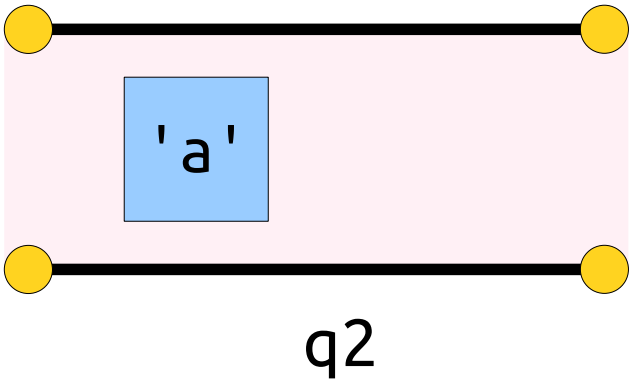
q2

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;
}
```

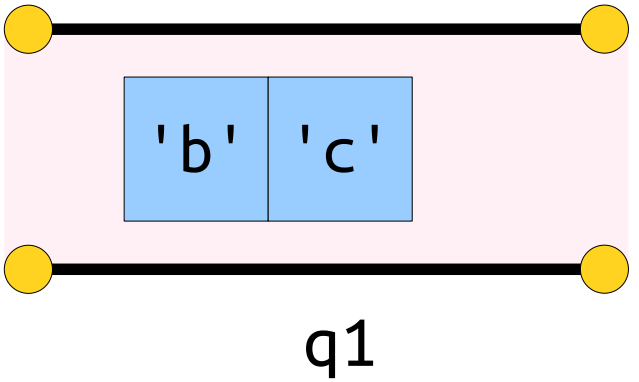


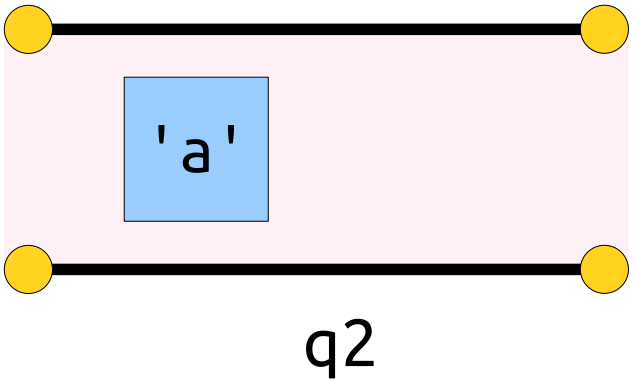


Queue

- 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;
}
```

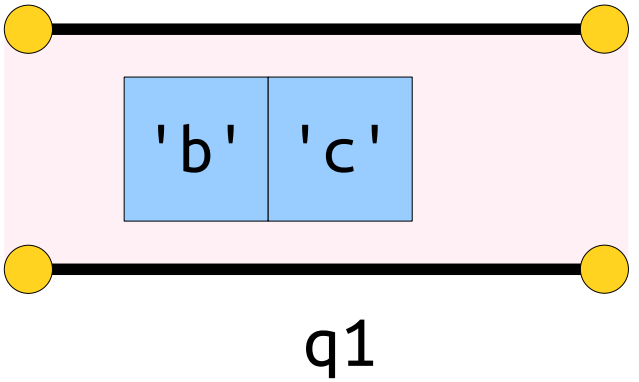


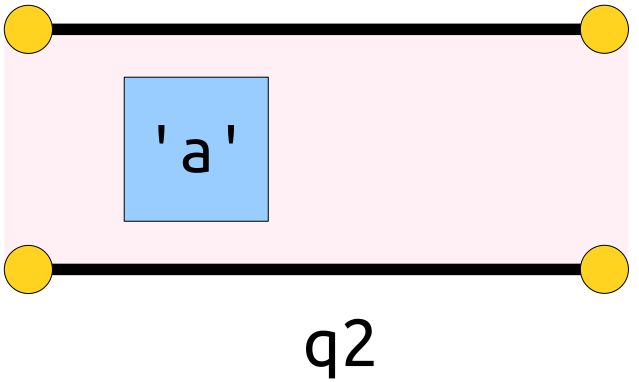


Queue

- 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;
}
```

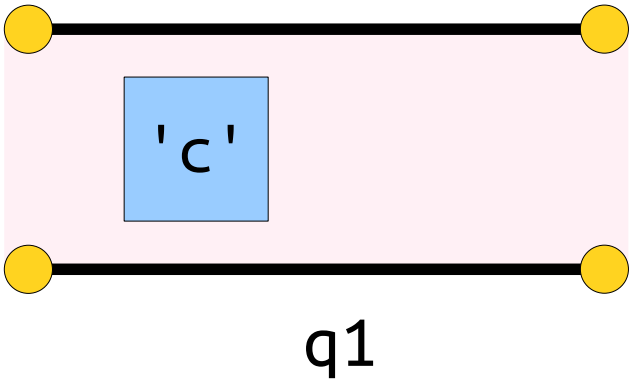


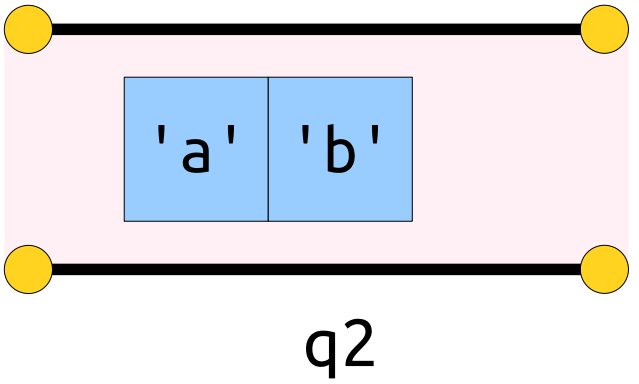


Queue

- 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;
}
```

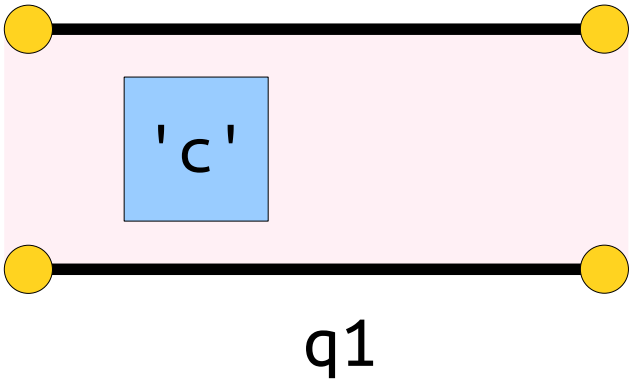


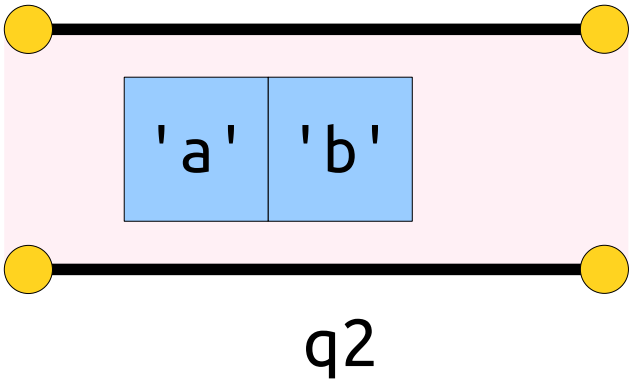


Queue

- 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;
}
```

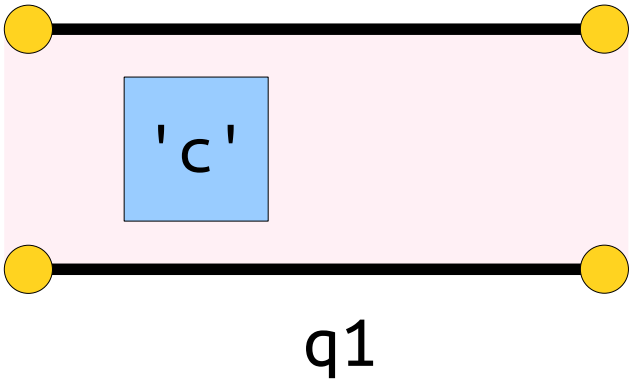


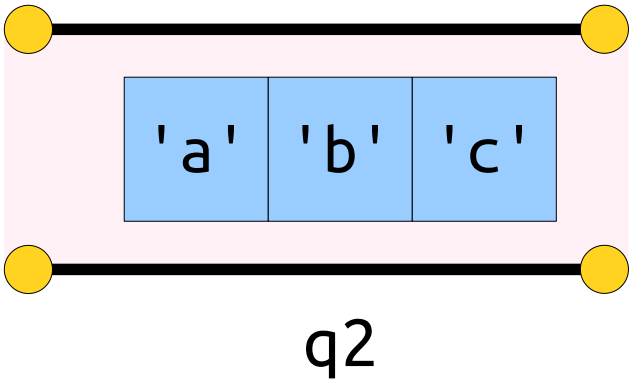


Queue

- 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;
}
```

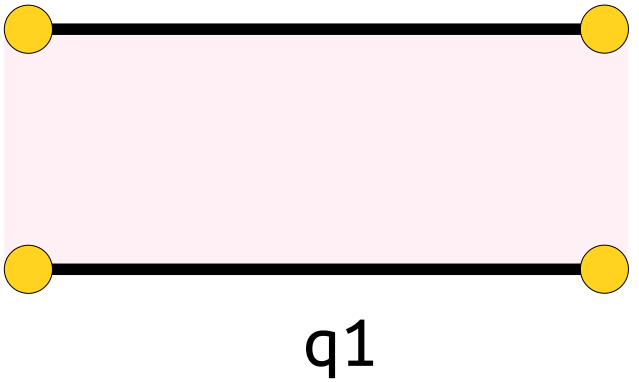


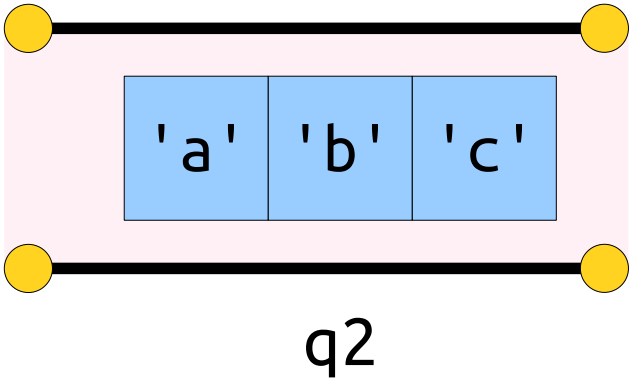


Queue

- 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;
}
```

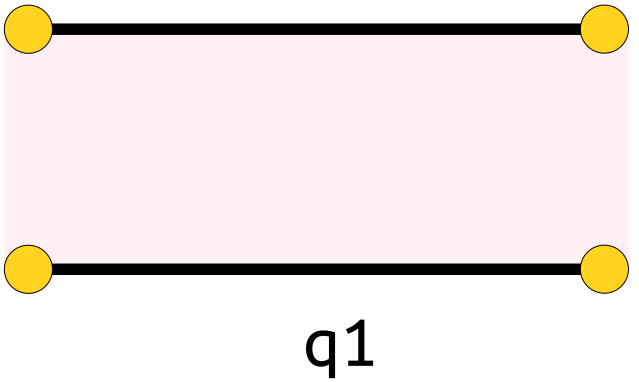


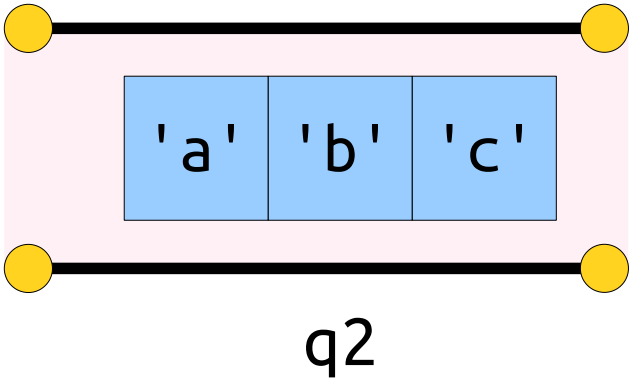


Queue

- 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;
}
```

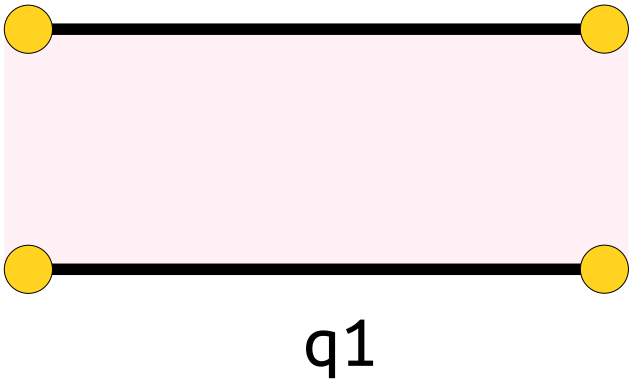


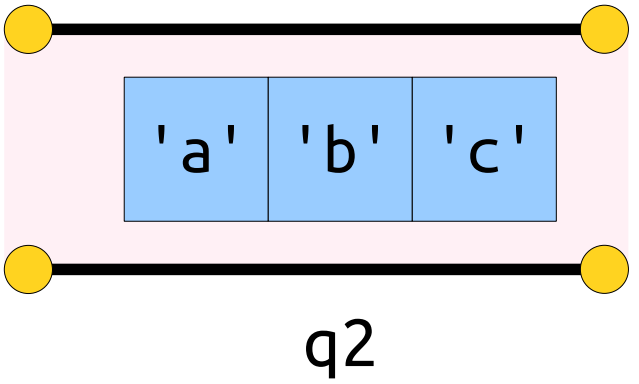


Queue

- 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;
}
```



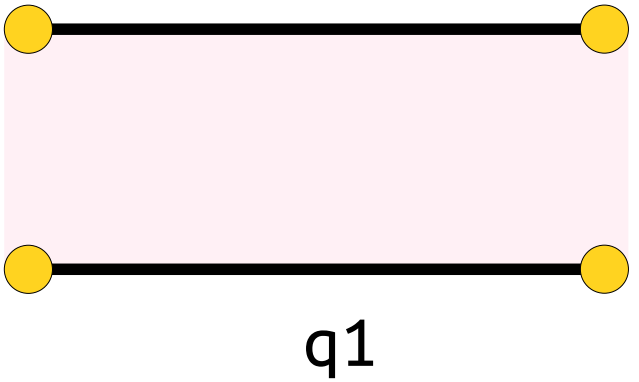


Queue

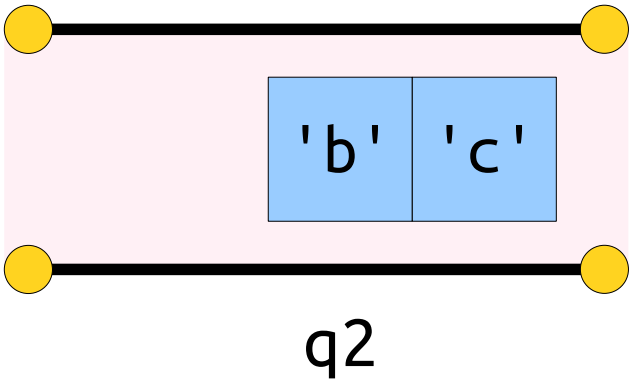
- What does this code print?

```
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



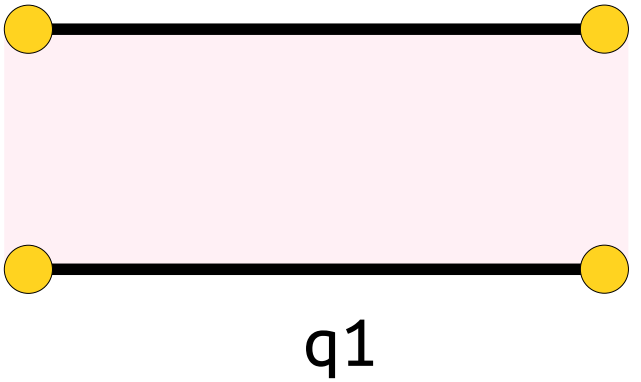
q2

Queue

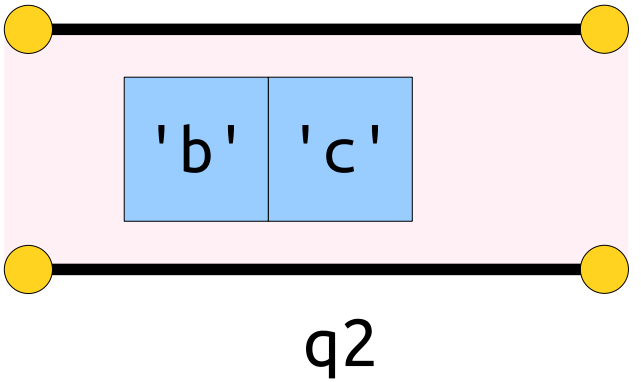
- What does this code print?

```
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

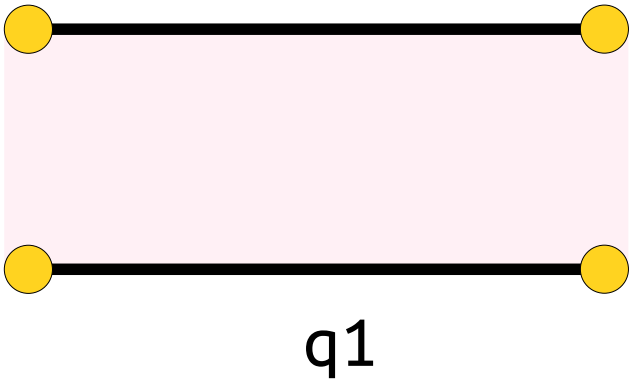


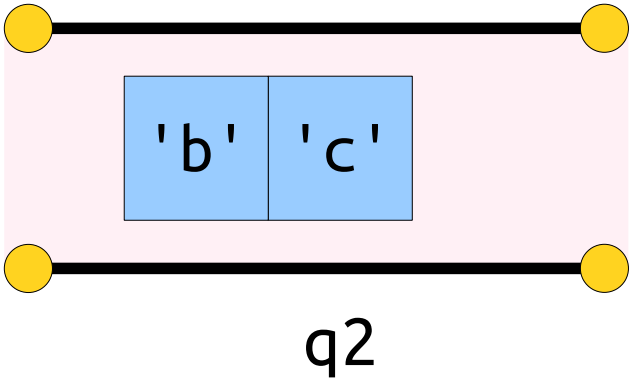
q2

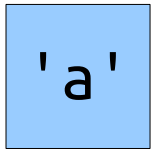
Queue

- 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;
}
```





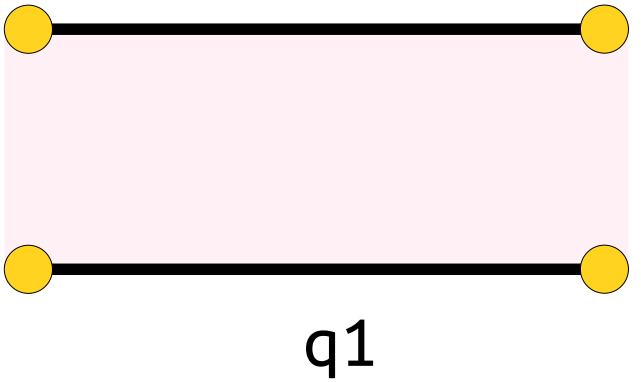


Queue

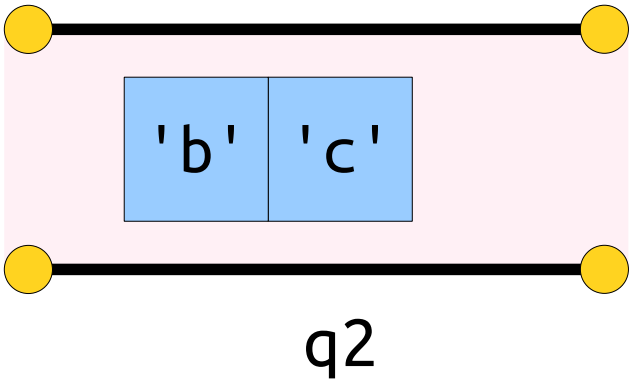
- What does this code print?

```
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

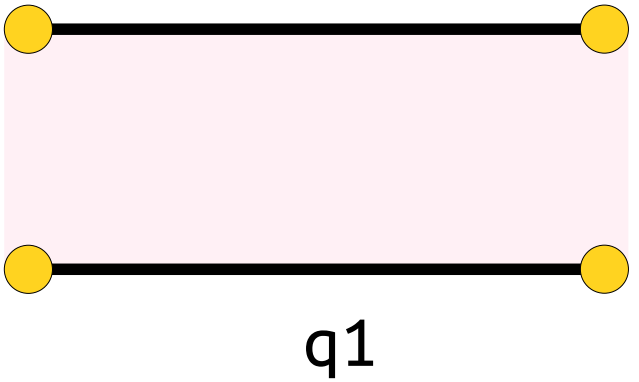


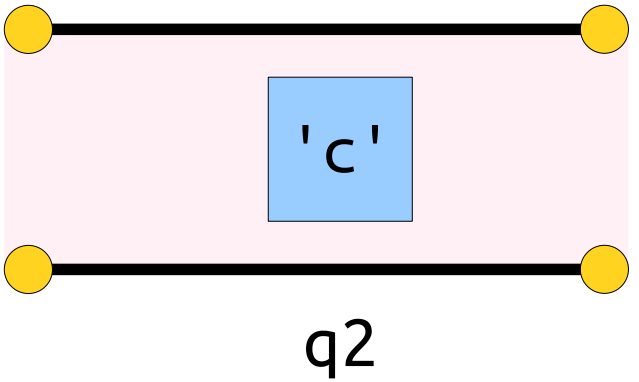
q2

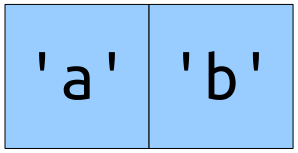
Queue

- 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;
}
```



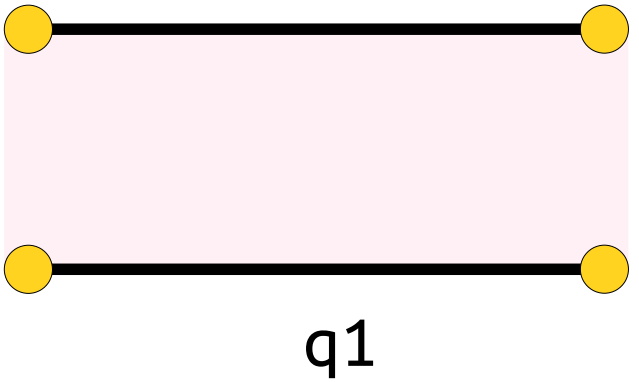


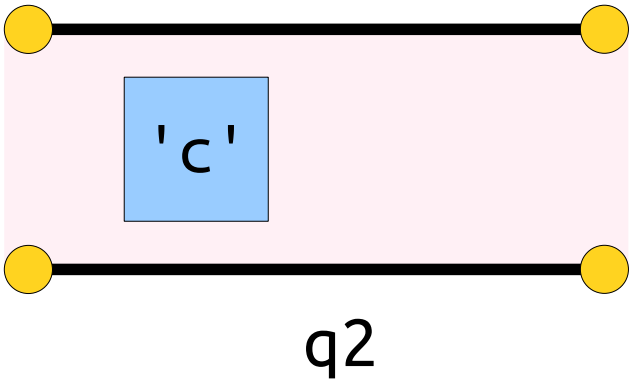


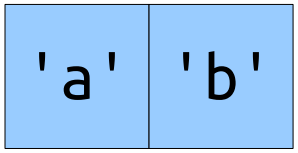
Queue

- 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;
}
```



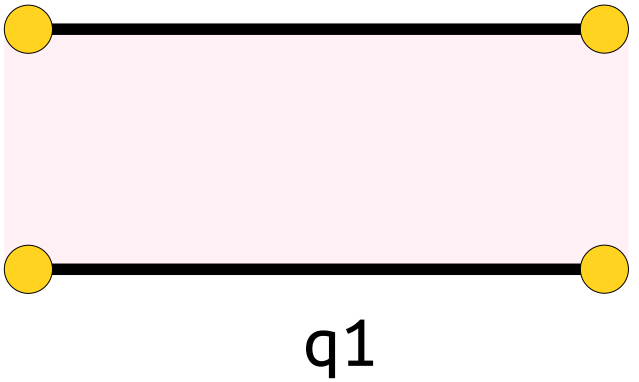


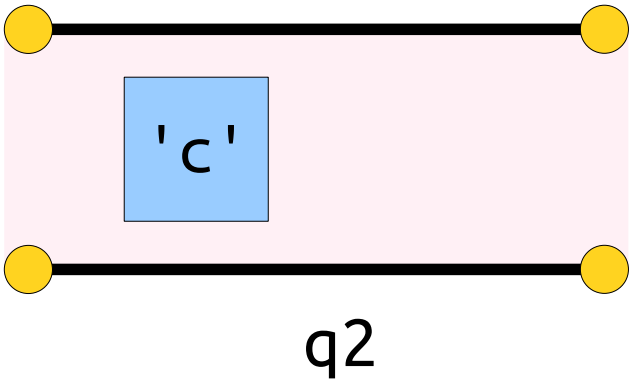


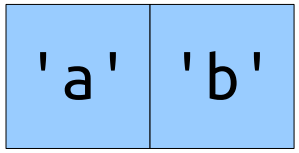
Queue

- 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;
}
```



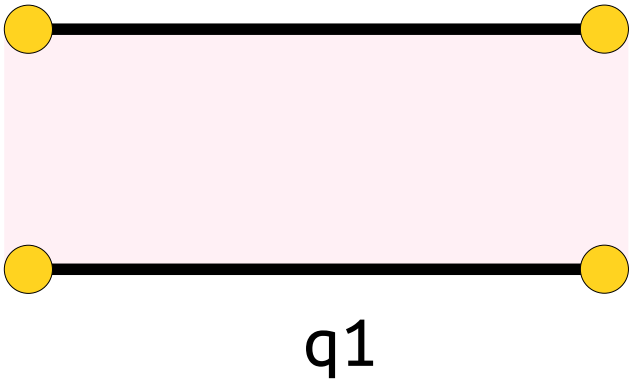


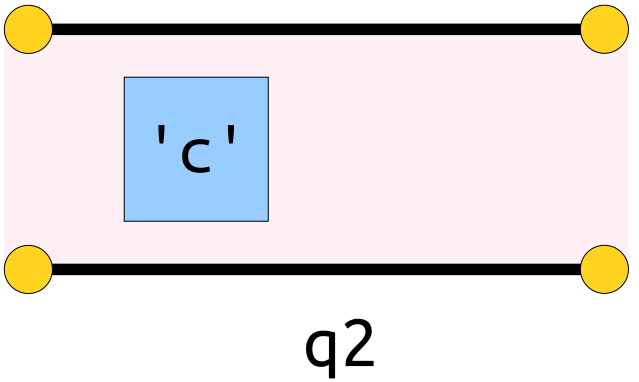


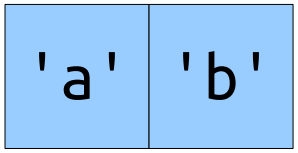
Queue

- 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;
}
```



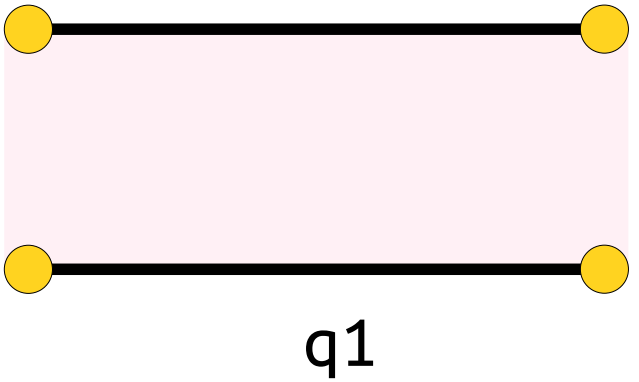


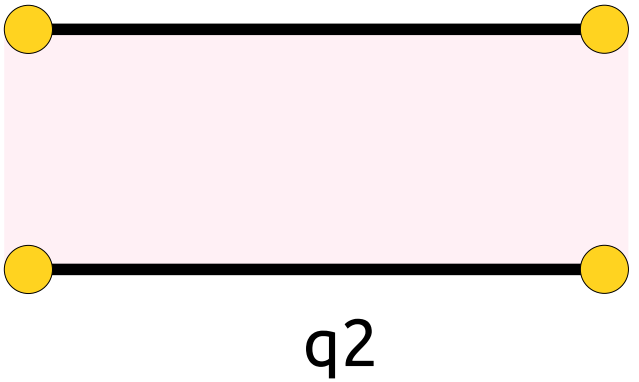


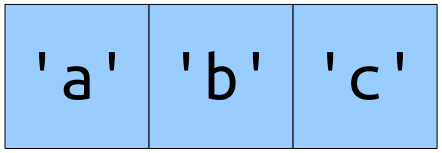
Queue

- 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;
}
```



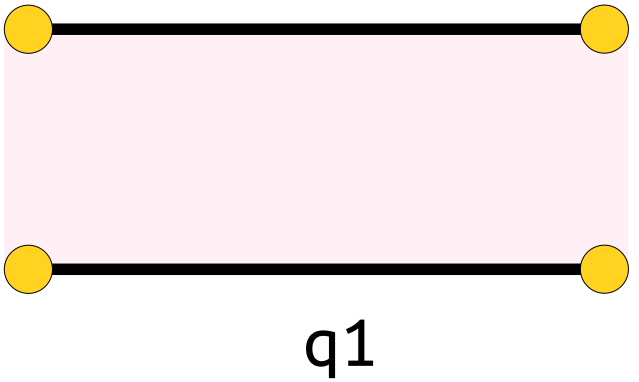


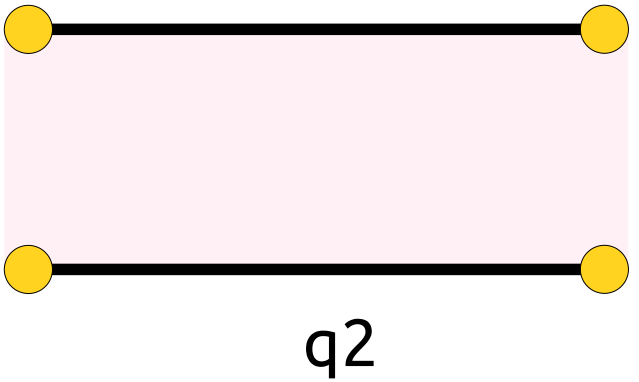


Queue

- 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;
}
```



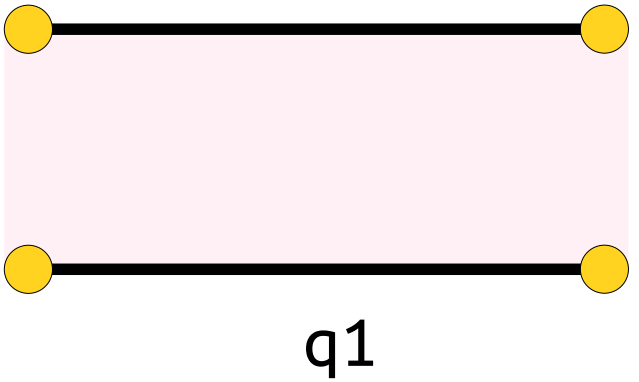


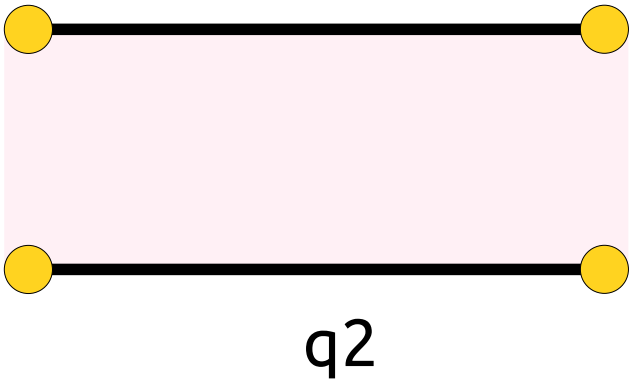
'a'	'b'	'c'

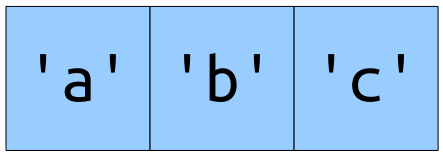
Queue

- 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.

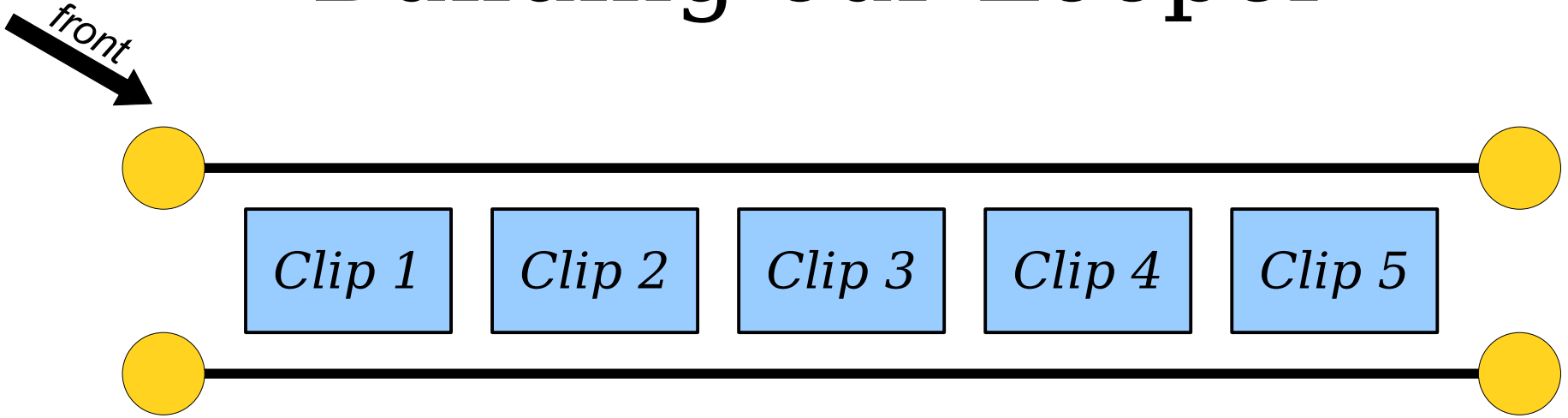
```
G2.wav 690
G2.wav 230
Bb2.wav 230
G2.wav 460
G2.wav 460
G2.wav 460
G2.wav 230
Bb2.wav 230
G2.wav 230
F2.wav 460
```

Building our Looper

Building our Looper

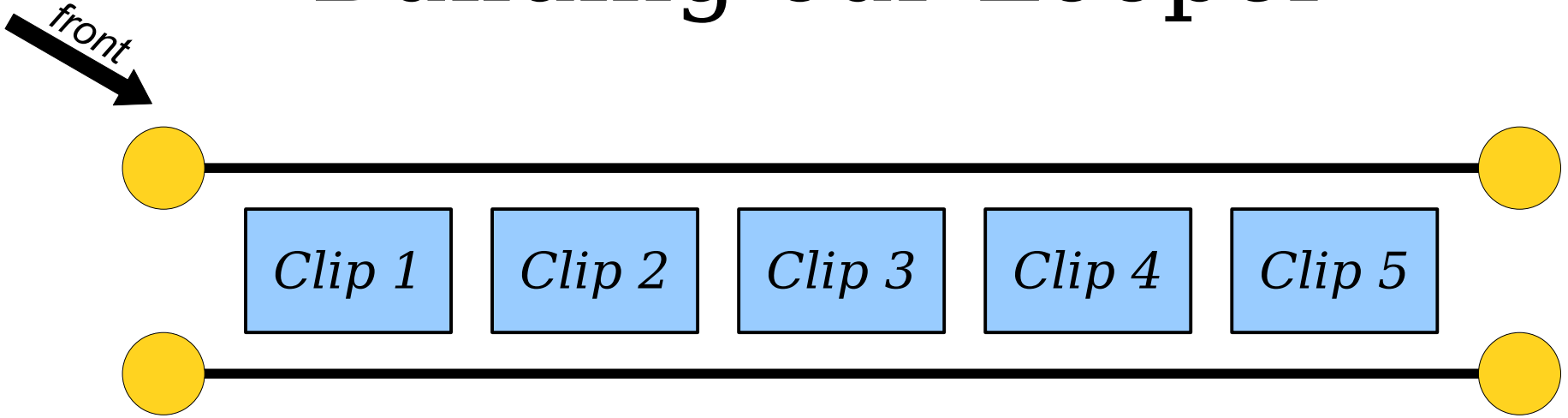
```
Queue<SoundClip> loop = loadLoop(/* … */);
```

Building our Looper



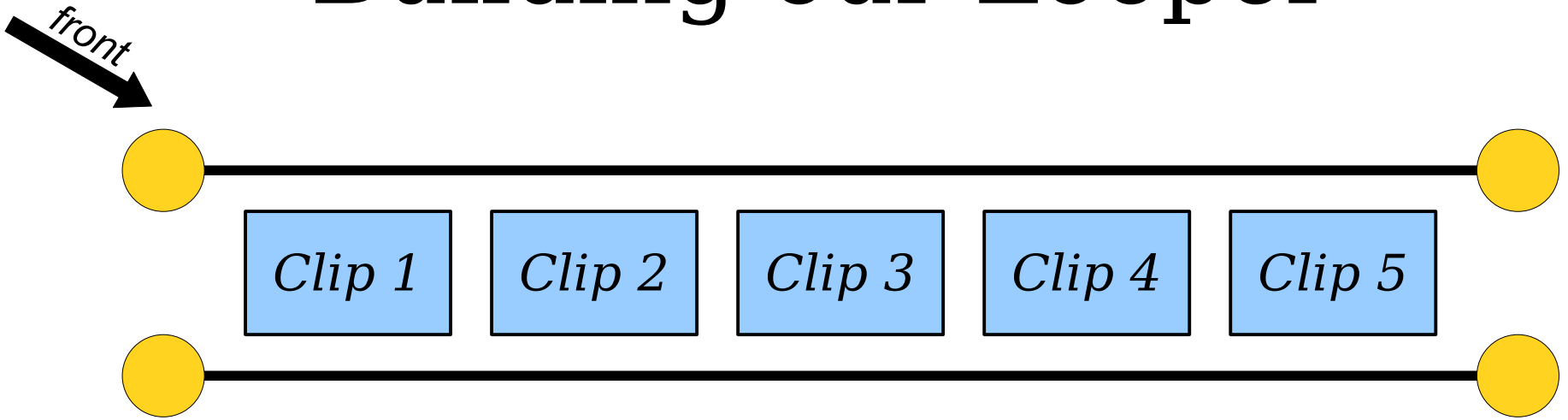

```
Queue<SoundClip> loop = loadLoop(/* … */);
```

Building our Looper



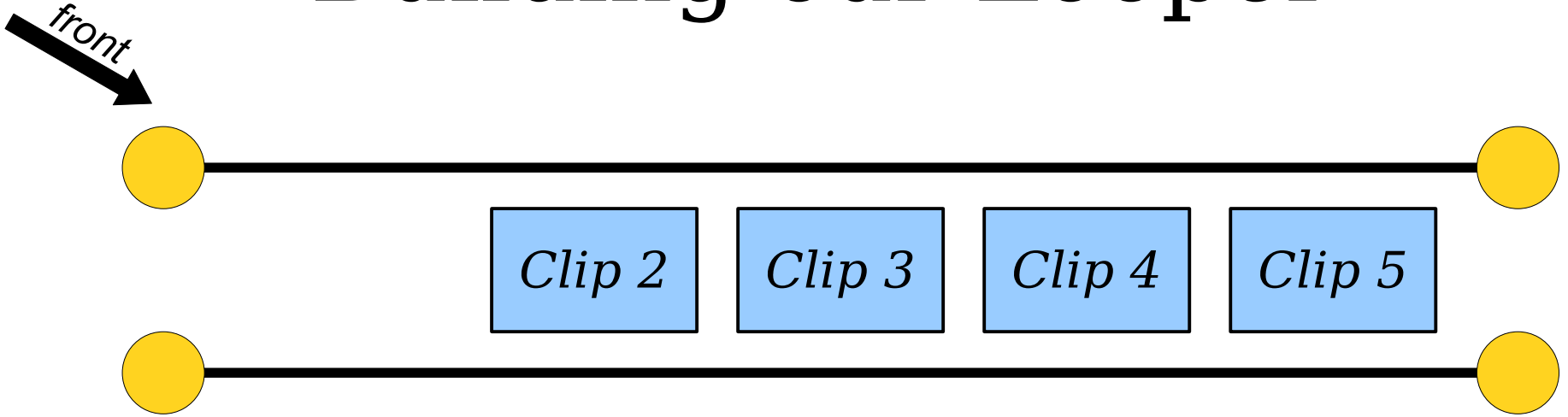

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {



}
```

Building our Looper




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



}
```

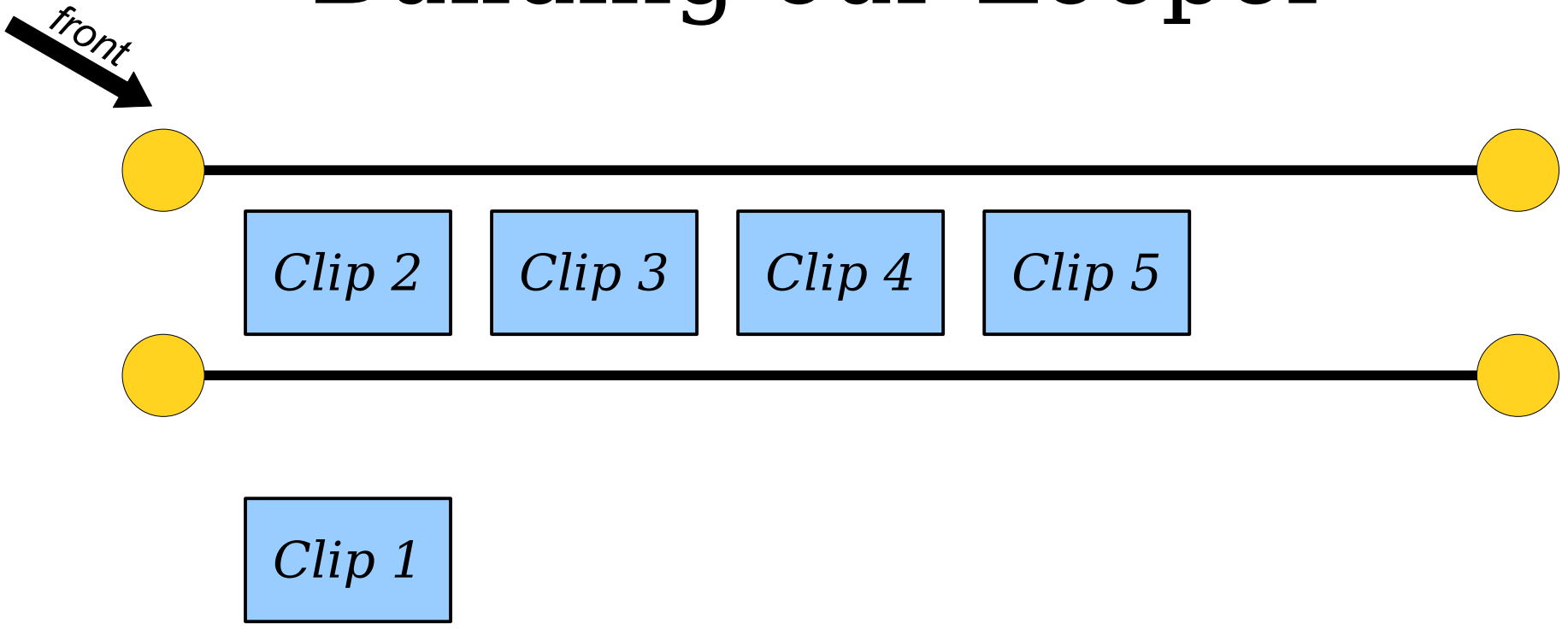
Building our Looper



Clip 1

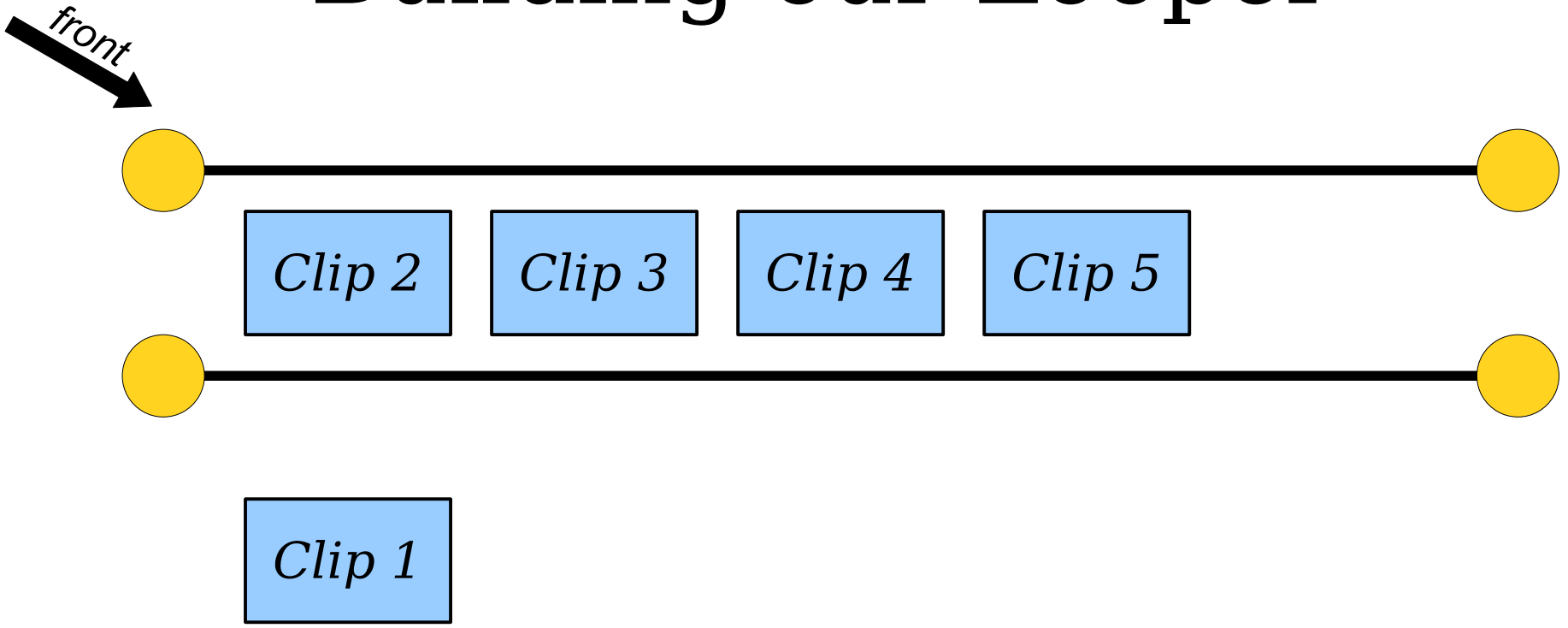
```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();



}
```

Building our Looper



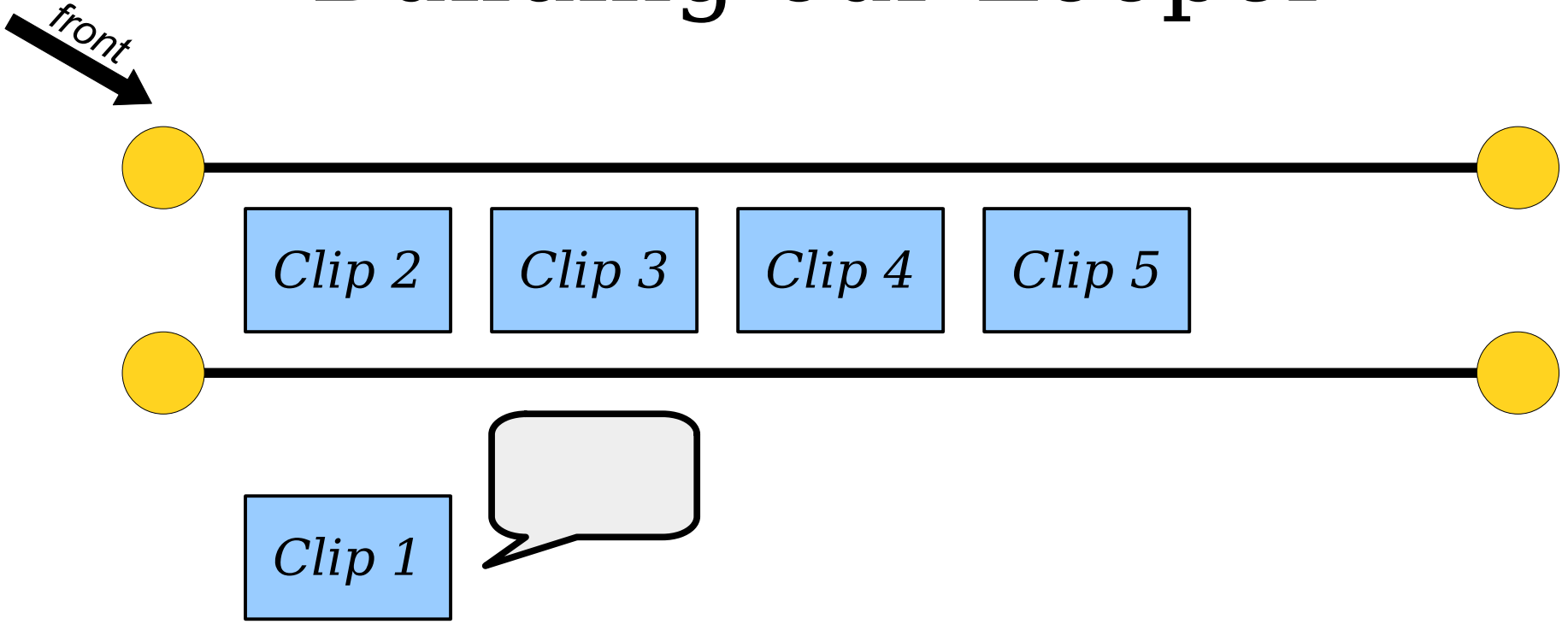

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();



}
```

Building our Looper



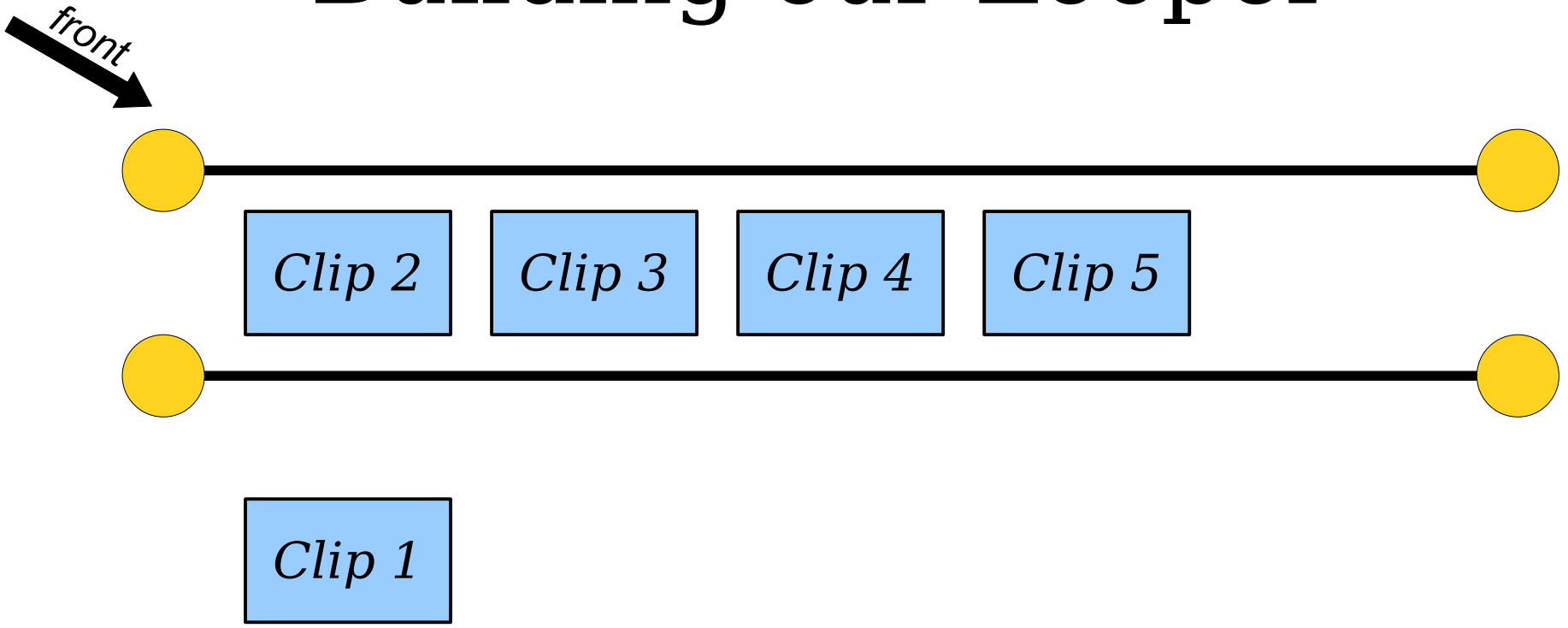

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

Building our Looper



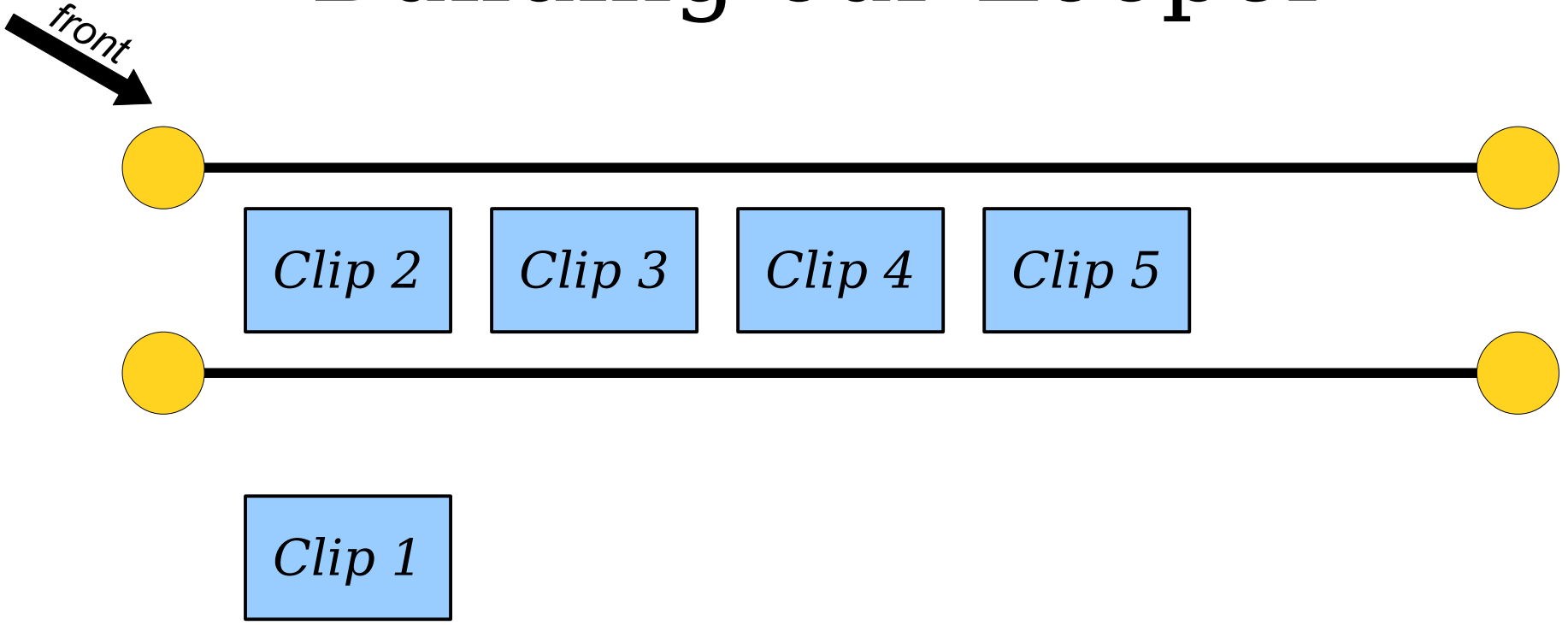

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

Building our Looper



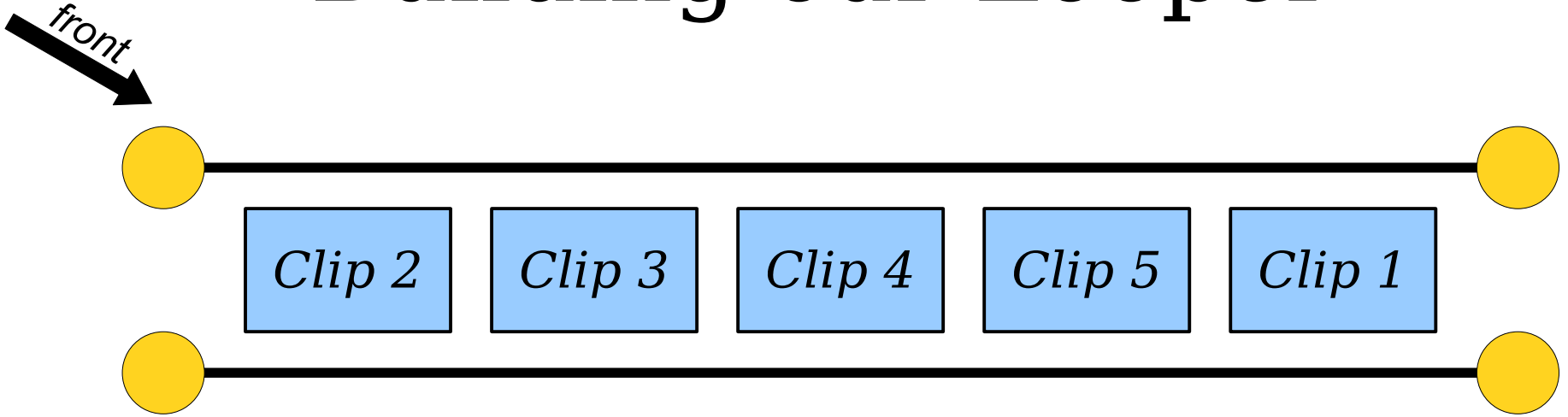

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);


}
```

Building our Looper



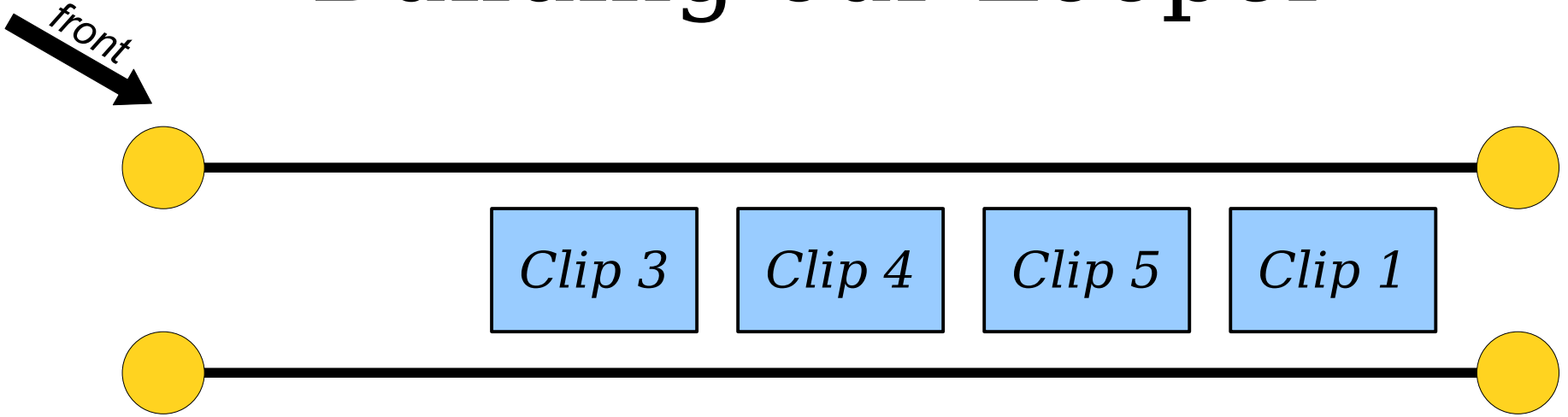

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper




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

  loop.enqueue(toPlay);
}
```

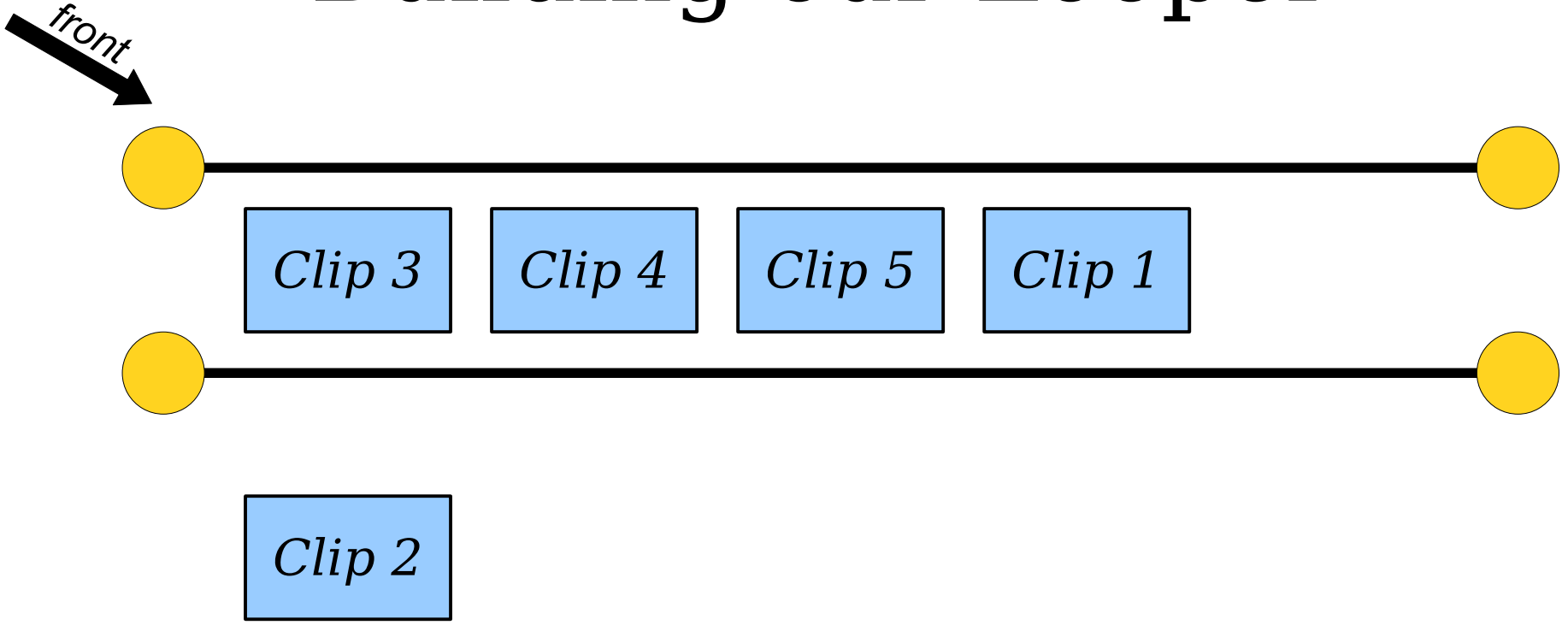
Building our Looper



Clip 2

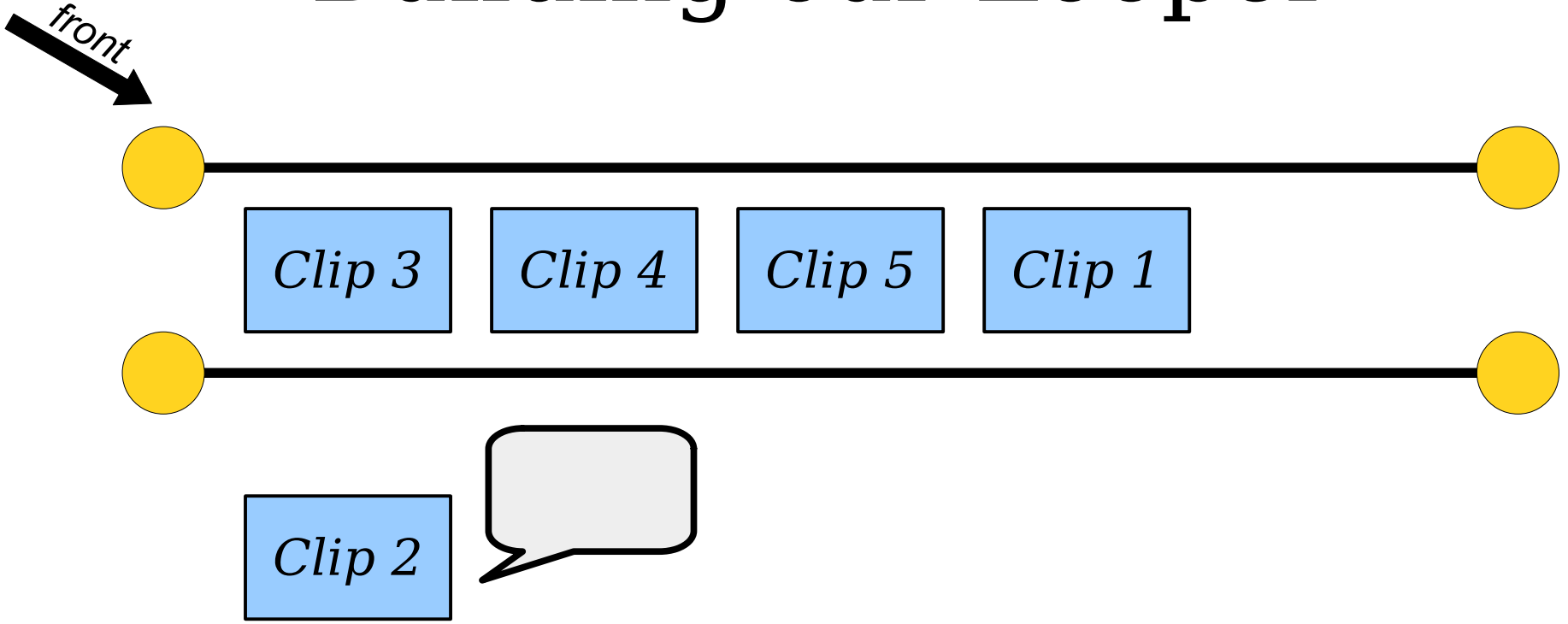
```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



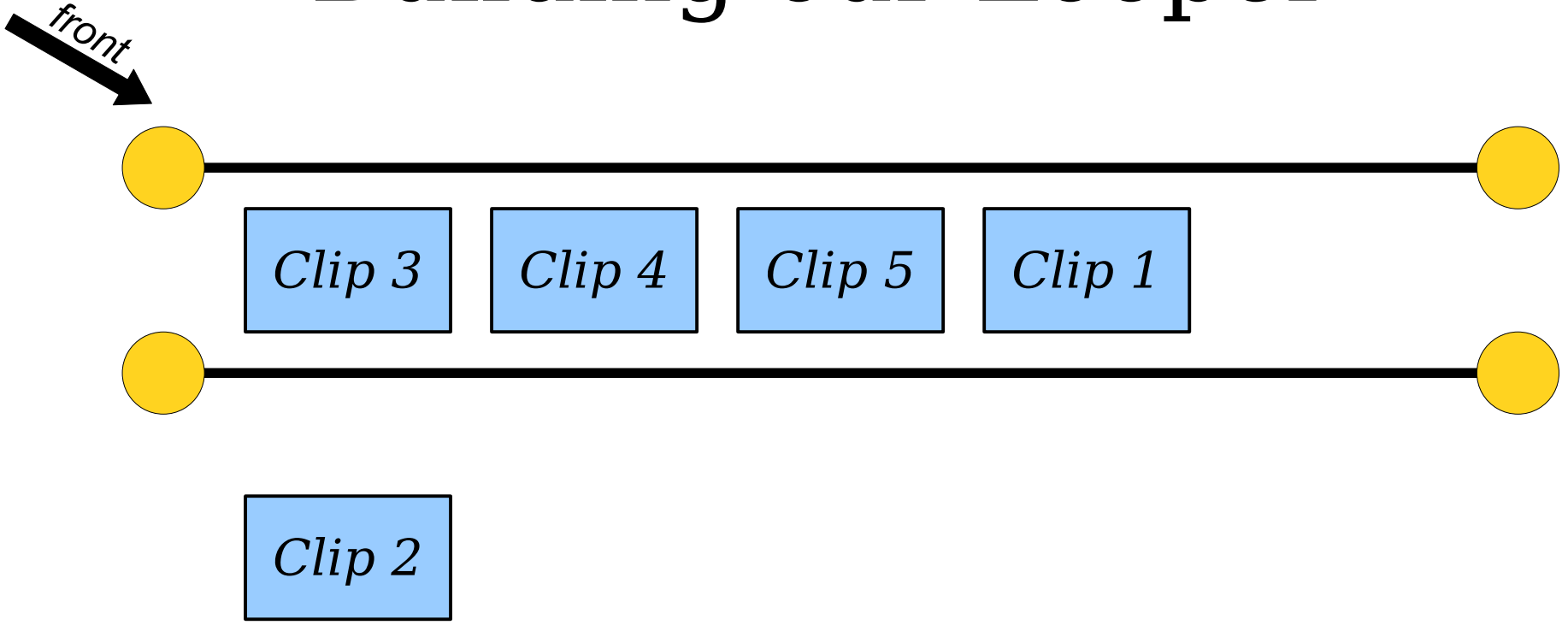

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



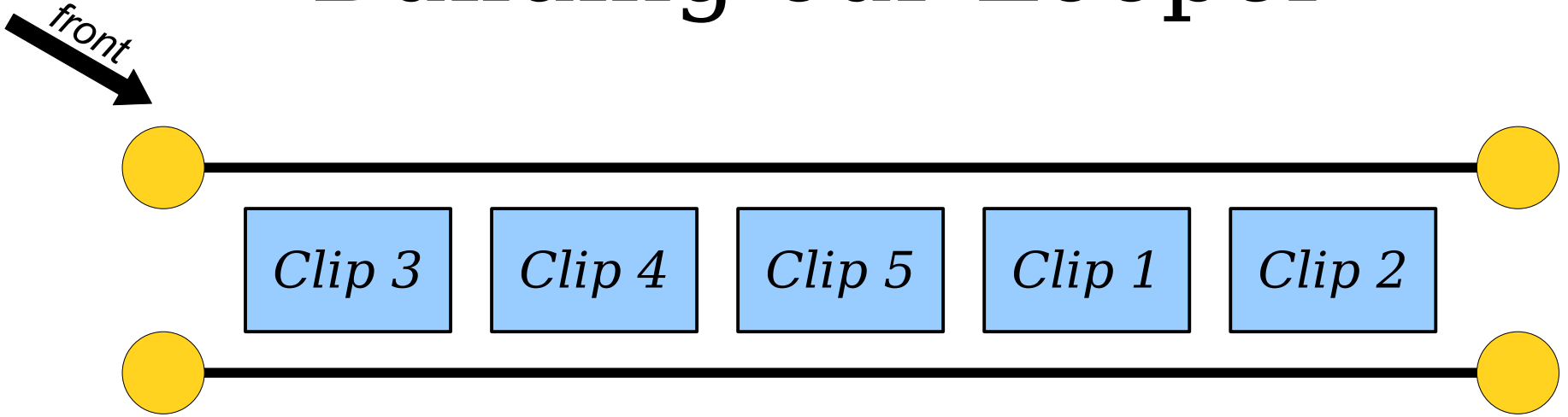

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



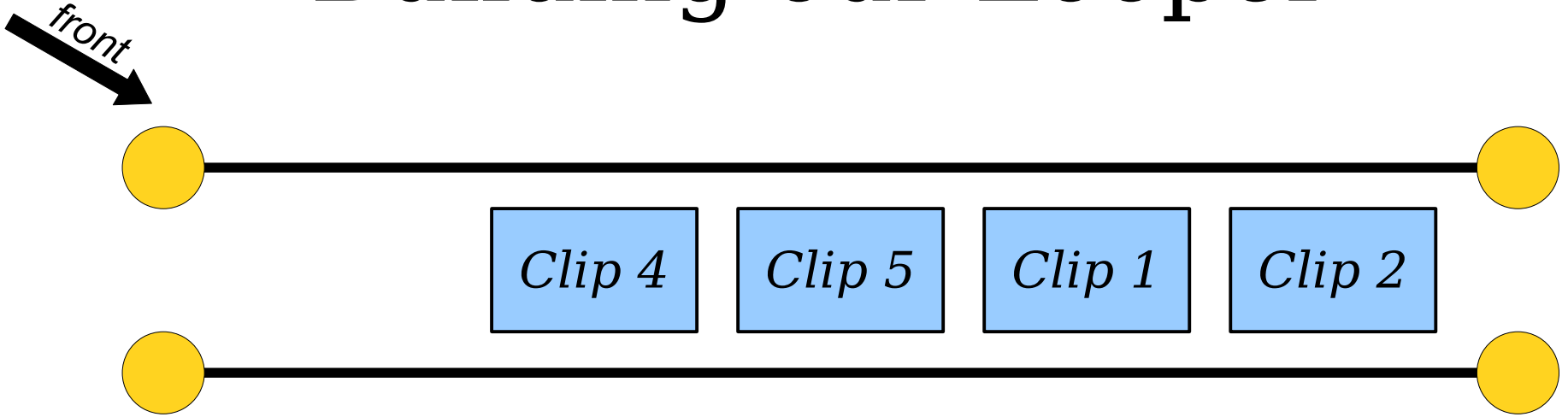

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper




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

  loop.enqueue(toPlay);
}
```

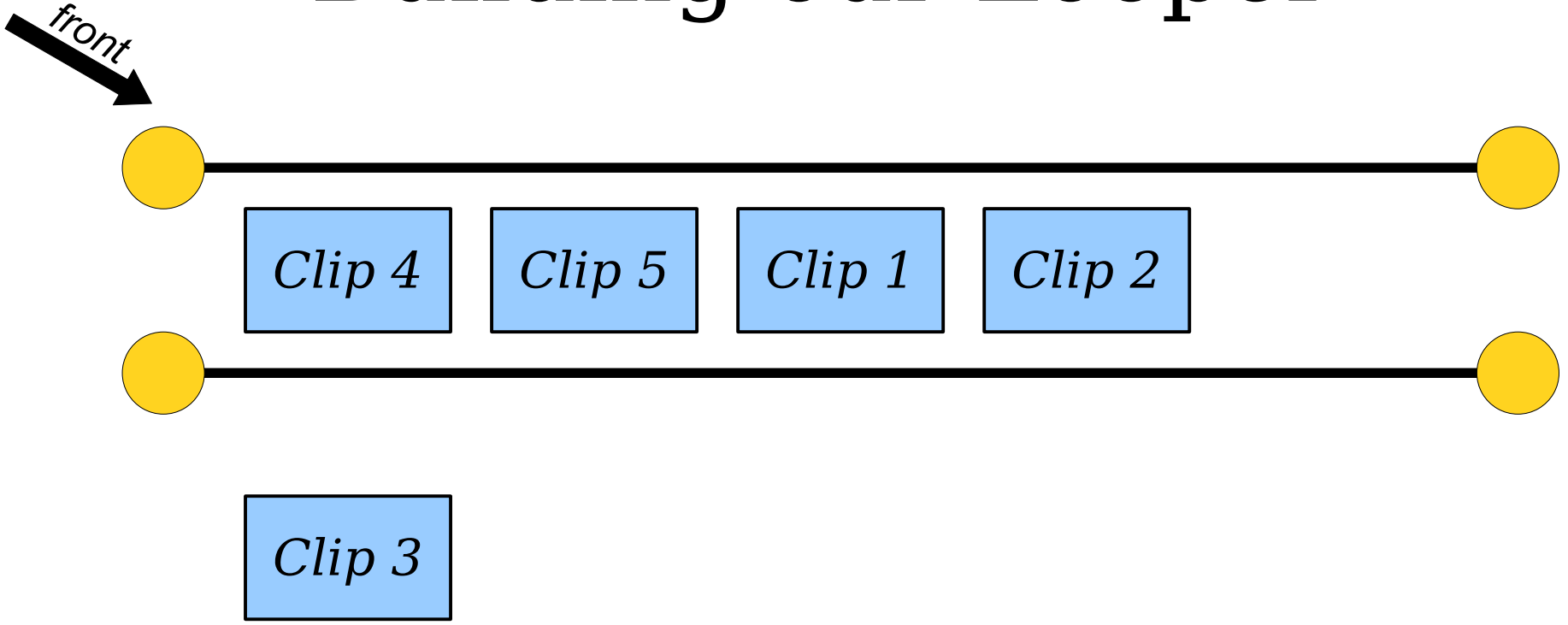
Building our Looper



Clip 3

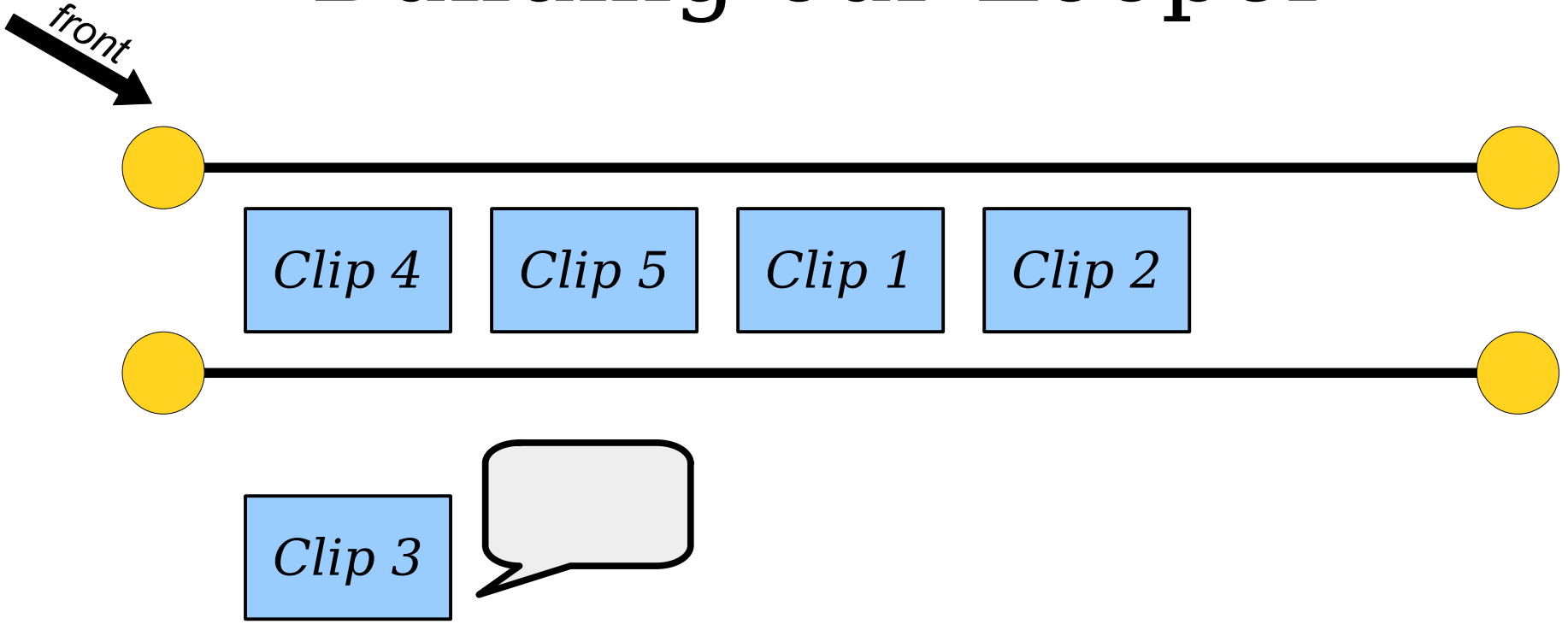
```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



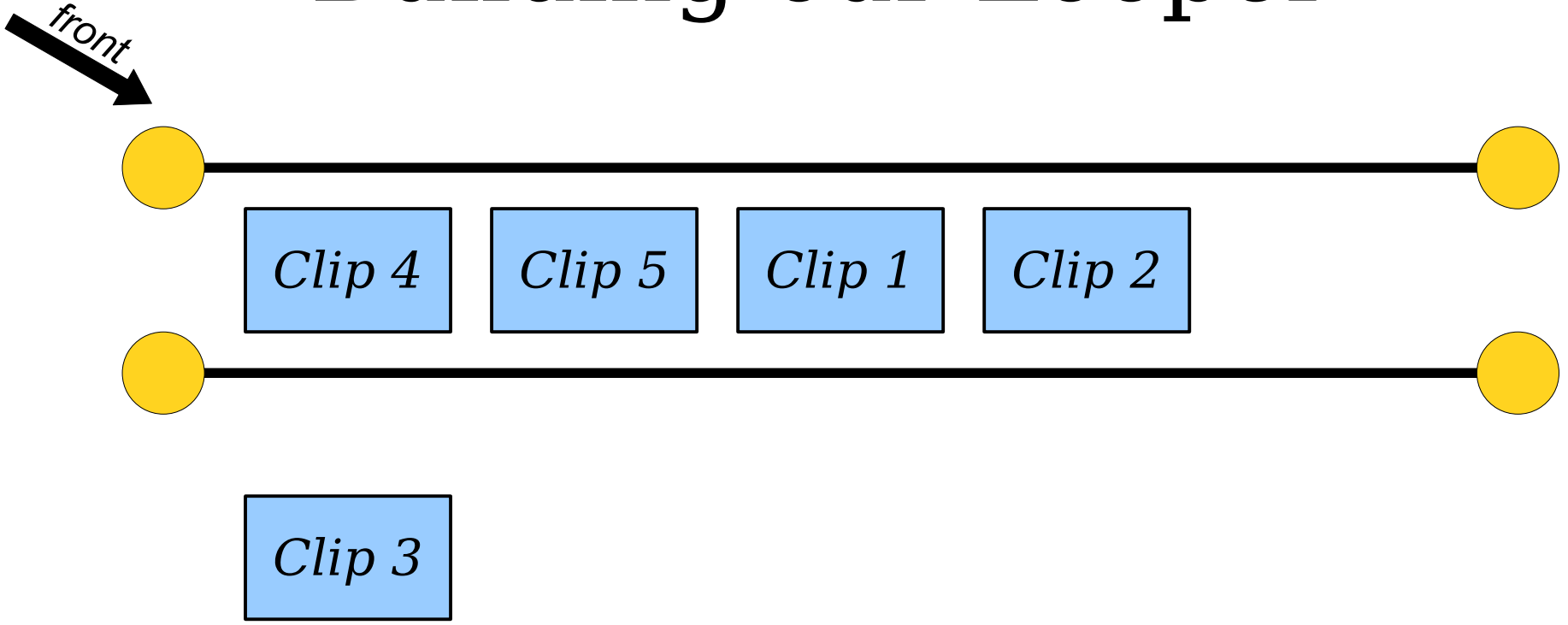

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  loop.enqueue(toPlay);
}
```

Building our Looper



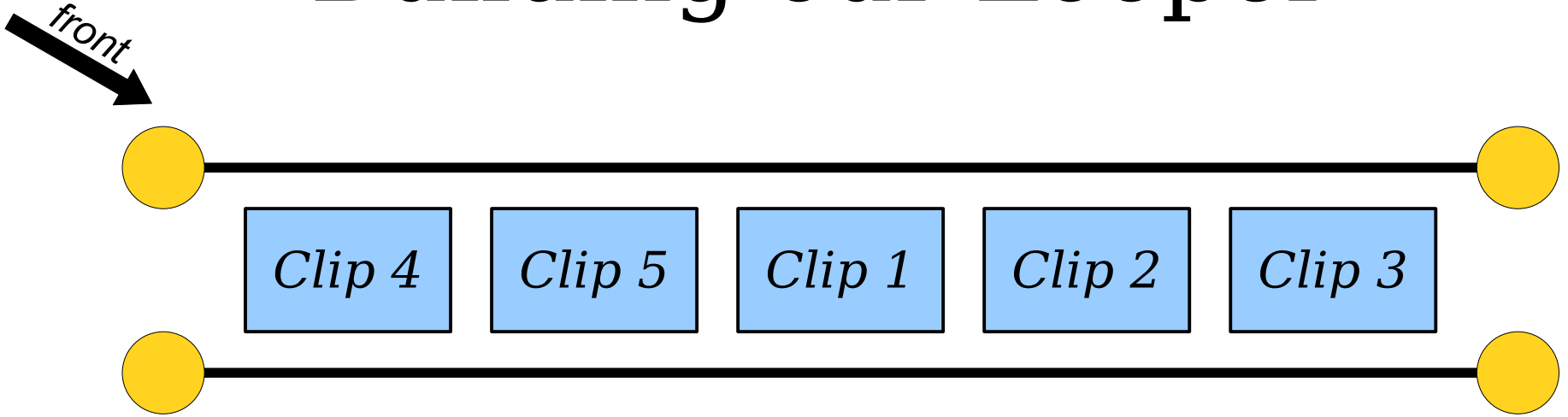

```
Queue<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.dequeue();
  playSound(toPlay.filename, toPlay.length);

  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);
}
```

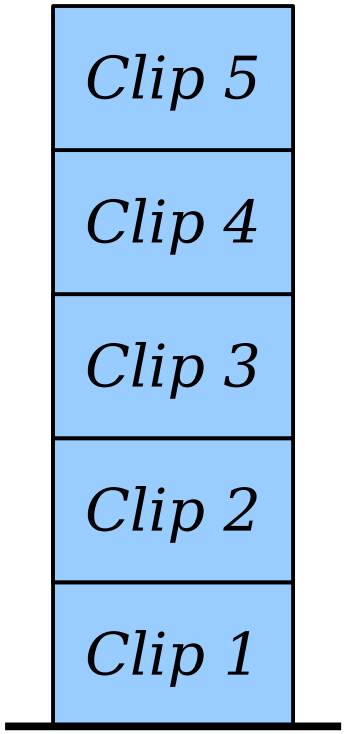
Changing our Looper

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

Answer at
https://pollev.com/cs106bwin23

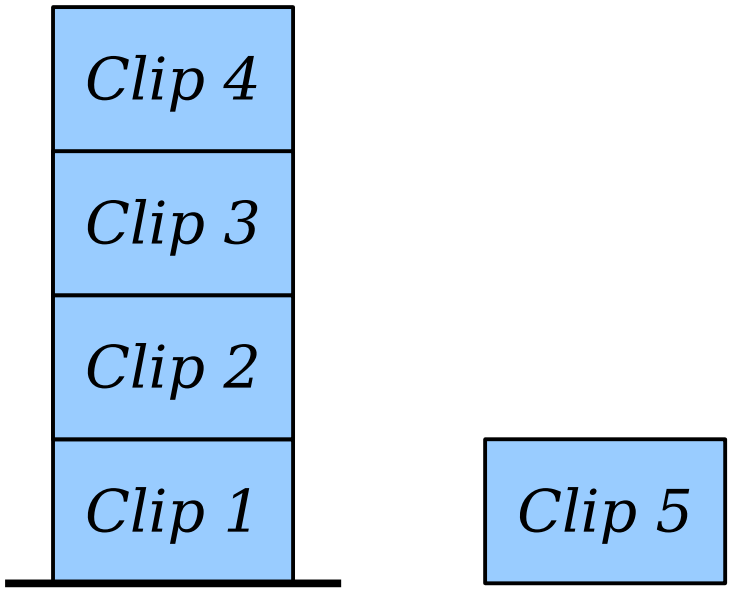
```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



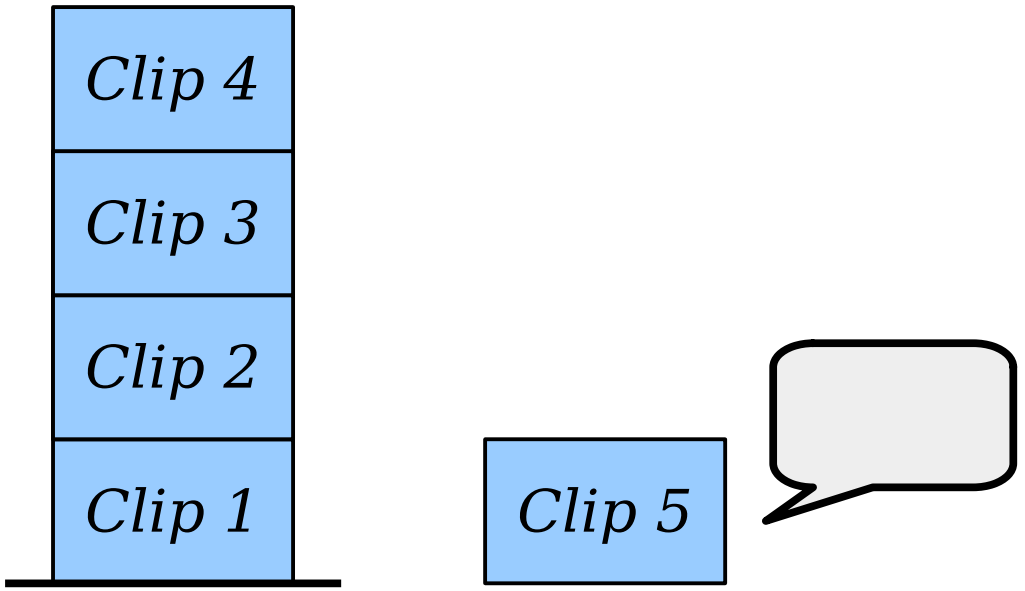

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



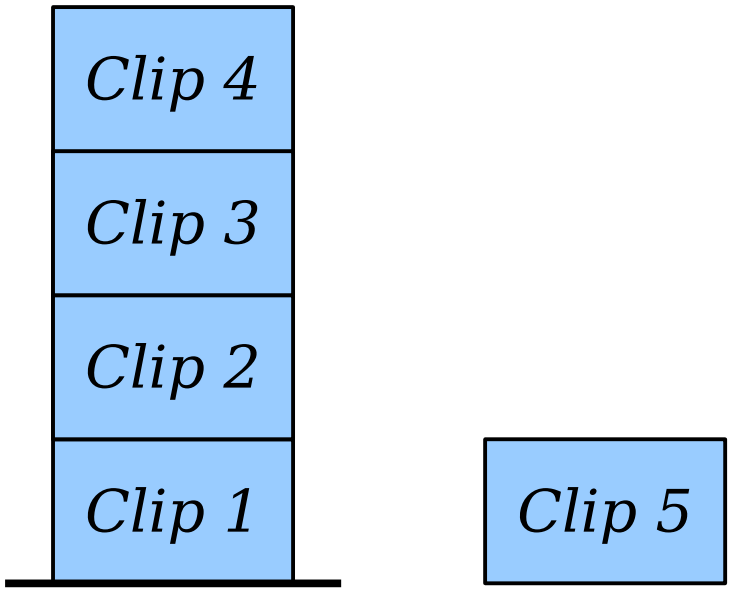

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



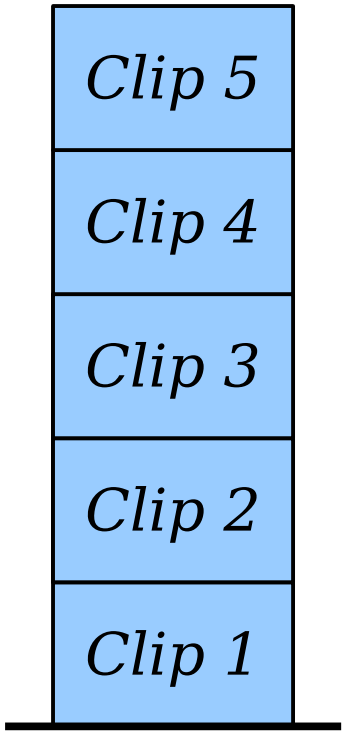

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



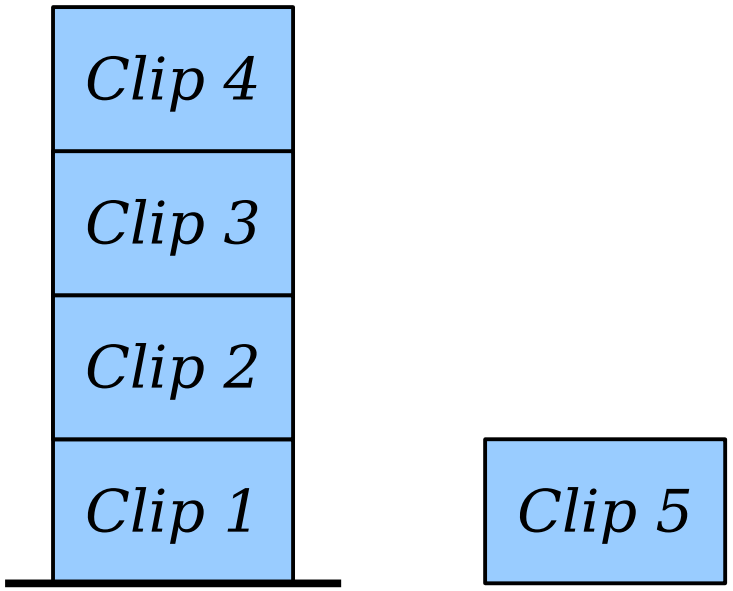

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



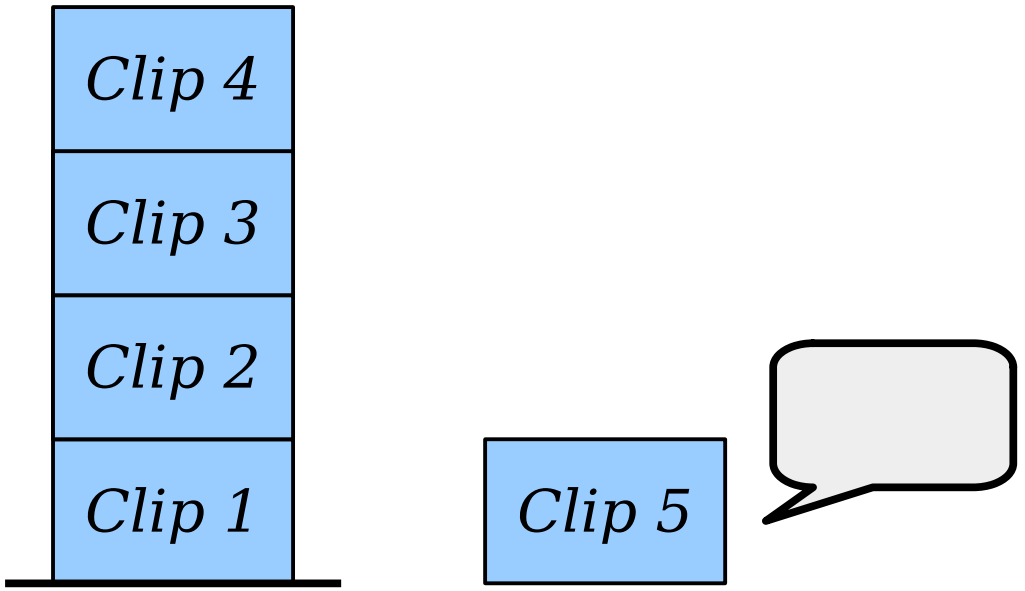

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



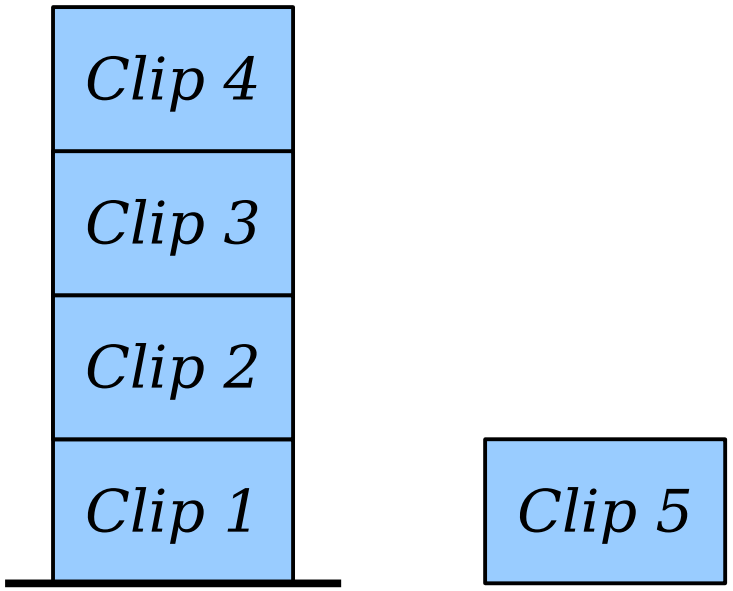

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  loop.push(toPlay);
}
```

Changing our Looper



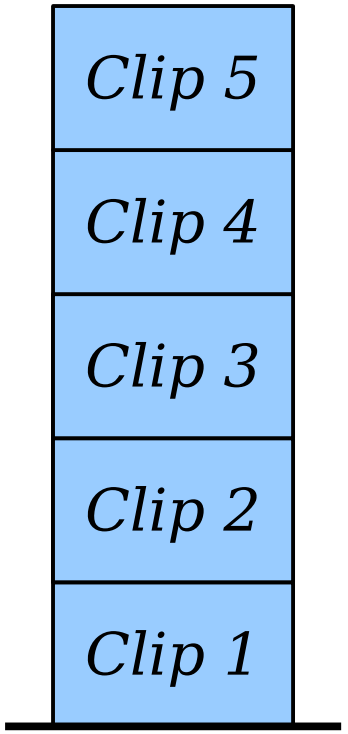

```
Stack<SoundClip> loop = loadLoop(/* … */);
while (true) {
  SoundClip toPlay = loop.pop();
  playSound(toPlay.filename, toPlay.length);

  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

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